高等学校计算机专业规划教材

带您启航Java学习的风帆

用连贯的案例配套理论知识点实践

提供完整Java代码及教学PPT下载

Java程序设计教程



肖云鹏 李暾 刘宴兵 编著

清华大学出版社

VISIT…

LANZAROTE
Caliente.COM

高等学校计算机专业规划教材
带您启航 Java 学习的风帆
用连贯的案例配套理论知识点实践
提供完整 Java 代码及教学 PPT 下载

Java 程序设计教程

肖云鹏　李　暾　刘宴兵　编著

清华大学出版社
北　京

内容简介

本书是为大学本专科Java程序设计课程准备的教材。全书以“what、why、how”的方式讲解，强调原理，重视实践。全书贯穿一个实例，把大学教学最常使用的“图书管理系统”作为实例，从第1章开始，安排在每一章的最后一节。纵向，各章承前启后，层层递进，从最简单的控制台、一个类图书管理系统→控制台、多个类的图书管理系统→合理的数据结构、代码设计的图书管理系统→带数据库的图书管理系统→有漂亮界面的图书管理系统→带网络连接的图书管理系统→带多线程、多客户端可以并行的图书管理系统。最后，将图书管理系统稍作修改，实现了一个简单QQ的程序。横向，对于每一章，最后一节的实例也是对本章学习内容的总结和实践。

同时，根据实际教学情况，我们在本书的实例中用最简单的方式融汇了面向对象、数据结构、数据库、网络编程、多线程、通信协议、程序结构、常用设计模式等同学们在前期课程中学习过，但在实际运用中不一定能掌握的重要知识点。为了配合教师教学及同学们自学，本书提供了配套教学的PPT和所有章节的源代码。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目(CIP)数据

Java程序设计教程/肖云鹏，李暾，刘宴兵编著. —北京：清华大学出版社，2019
(高等学校计算机专业规划教材)
ISBN 978-7-302-52990-3

Ⅰ. ①J…　Ⅱ. ①肖…　②李…　③刘…　Ⅲ. ①JAVA语言－程序设计－高等学校－教材　Ⅳ. ①TP312.8

中国版本图书馆CIP数据核字(2019)第093907号

责任编辑：贾　斌
封面设计：何凤霞
责任校对：徐俊伟
责任印制：杨　艳

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 装 者：北京嘉实印刷有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：14.25　　**字　　数**：335千字
版　　次：2019年9月第1版　　**印　　次**：2019年9月第1次印刷
印　　数：1～1500
定　　价：39.80元

产品编号：073416-01



前言

本书是为大学本专科 Java 程序设计课程准备的教材。本书的编写受到我上一本教材《Android 程序设计教程》的激励。没想到上一本教材出版后，会有那么多高校和同学一直在使用，这给了我很大的激励，于是鼓起勇气坚持就我负责的 Java 系列另外两门课写出相应的教材，即《Java 程序设计教程》《Java Web 程序设计教程》，希望能够帮到更多的同学。笔者总结了多年来的教学和工程经验，力争使本教材达到以下学习目标。

- 在每一个重要的知识点上，以"what、why、how"的方式讲解。在讲是什么(what)问题的时候，多做比喻、多讲故事、多画图。让同学们首先有感性认识，再落脚到程序代码层面，让学习的过程从感性认识到理性认识再到量化实现。在讲原理(why)的时候，尽量深入透彻，这是对于同学们非常重要的要求。我常和我的学生们说，清楚原理才能做出优秀的程序。最后落实到 how 的问题，即使用的问题。
- 全书贯穿一个实例，把大学教学最常使用的"图书管理系统"作为实例，从第 1 章开始，安排在每一章的最后一节。纵向，各章承前启后，层层递进，从最简单的控制台、一个类图书管理系统→控制台、多个类的图书管理系统→合理的数据结构、代码设计的图书管理系统→带数据库的图书管理系统→有漂亮界面的图书管理系统→带网络连接的图书管理系统→带多线程、多客户端可以并行的图书管理系统。最后，将图书管理系统稍作修改，实现了一个简单的 QQ 程序。横向，对于每一章，最后一节的实例也是对本章学习内容的总结和实践。
- 我们根据多年来教学经验，针对教学中学生实际存在的问题，在本书的实例中用最简单的方式融汇了面向对象、数据结构、数据库、网络编程、多线程、通信协议、程序结构、常用设计模式等同学们在前期课程中学习过，但在实际运用中不一定能掌握的重要知识点。我们的初衷是希望本书不仅仅是一门 Java 程序的教材，更希望同学们通过这门课程的学习，对整个本科阶段的重要课程进行整理，以点带面，启发同学们的学习热情。如在网络编程一章，我们首先从几个最基本的网络问题讲起。这样做的目的是尽可能地深入浅出，融会贯通，同时保证大部分几乎零基础的同学都能学会。这种编写方式也是我们在实际教学中采用的授课方式。

- 本书主要作为本科教材,因此这不是一本很厚的、面面俱到的Java书,而且我们认为本科的教学本身也应该是启发式的教学。我在课堂上常要求同学们大学期间在专业课学习上做到三点:(1)扎实的专业基础知识;(2)良好的英文读写水平;(3)快速掌握陌生知识。通过课堂上有限授课时间和学期内有限的课程学习,同学们打好基础,掌握学习方法,相信有兴趣的同学自然会“自学长才”,我想这也是大学学习的要领。也是基于这个想法,这本教材讲到的知识都是最重要、最基础的问题,因此在书中没有要求Java版本问题。
- 为了配合教学和同学们自学,本书提供了配套教学的PPT和所有章节的源代码。读者可到清华大学出版社网站进行下载。

本书的写作工作是由我们实验室三位老师共同完成的,几位作者都是长期在Java领域从事理论教学、工程实践、项目合作的老师。我们的想法是通过我们的努力,以开放的心态,能够帮助更多的希望学习Java的同学。

整个书稿从开始有创作想法到最后出版,前前后后修改了几十稿,光我们实验室打印机的墨盒就换了两块,反复打印、修改。写书真的既是技术活又是体力活,把上课时讲授的知识点变成文字是个巨大的工程。诚然,即便是我们非常努力完善书稿,但由于我们水平有限和时间仓促,本书可能还会有这样或那样的问题,恳请读者批评指正。我们希望本书的第二版、第三版等不仅是内容的更新,还会加入更多有趣的知识点。

最后,感谢我的家人对我工作的支持,感谢实验室的前辈、同事对我工作一直给力地支持,感谢实验室的同事和我一起讨论、拼搏的美好时光。

本书的完成得到重庆市重点研发项目(No. cstc2017zdcy-zdyfx0002,cstc2017zdcy-zdyfx0092)、重庆市基础科学与前沿技术研究项目(No. cstc2017jcyjAX0099)和重庆研究生教育教学改革研究项目(No. No. yjg183081)资助。

编　者

2019年4月



第 1 章 Java 入门 …… 1

1.1 本章任务 …… 1
1.2 Java 的故事 …… 1
1.3 三个版本的过去和现在 …… 4
1.4 环境搭建 …… 4
1.5 从控制台到 Eclipse——一个最简单的图书管理系统 V1.0 …… 6
1.6 一个简单的面向对象的改造 …… 10
1.7 JDK、JRE 和 JVM …… 11
1.8 养成良好的学习习惯 …… 12
1.9 如何导入本书的案例库 …… 12

第 2 章 基础知识 …… 14

2.1 本章任务 …… 14
2.2 Java 基本程序结构 …… 14
2.3 Java 程序基本代码规范 …… 15
2.3.1 注释规范 …… 16
2.3.2 标识符 …… 17
2.3.3 关键字 …… 17
2.4 数据类型与变量 …… 22
2.4.1 整型 …… 22
2.4.2 浮点型 …… 23
2.4.3 char 型 …… 24
2.4.4 布尔型 …… 24
2.4.5 变量 …… 25
2.4.6 常量 …… 25
2.5 运算符、表达式与控制语句 …… 26
2.5.1 运算符 …… 26
2.5.2 关系运算符 …… 27
2.5.3 逻辑运算符 …… 27
2.5.4 赋值运算符 …… 28



2.5.5 运算符优先级 …… 29
2.5.6 控制语句 …… 29
2.6 数组 …… 35
2.7 基本输入输出 …… 37
2.8 一个单机版、控制台、只有一个类的图书管理系统 V2.0 …… 39
2.8.1 运行效果图 …… 39
2.8.2 类结构示意图 …… 39
2.8.3 代码实现 …… 40

第3章 面向对象 …… 43

3.1 本章任务 …… 43
3.2 面向对象基本概念 …… 43
3.3 类与对象 …… 44
3.3.1 类 …… 44
3.3.2 对象 …… 45
3.4 封装 …… 45
3.5 继承 …… 47
3.6 多态 …… 47
3.7 抽象类和接口 …… 49
3.7.1 抽象类 …… 49
3.7.2 接口 …… 49
3.8 访问控制 …… 50
3.9 异常 …… 51
3.9.1 什么是异常 …… 51
3.9.2 异常处理 …… 51
3.10 三个常见的关键字 static、final、this …… 53
3.10.1 static 关键字 …… 53
3.10.2 final 关键字 …… 54
3.10.3 this 关键字 …… 55
3.11 图书管理系统 V3.0 …… 55
3.11.1 运行效果图 …… 55
3.11.2 类结构示意图 …… 55
3.11.3 代码实现 …… 56

第4章 集合 …… 60

4.1 本章任务 …… 60
4.2 集合——数据结构 Java 实现 …… 60
4.3 Java 集合的整体框架 …… 60



4.4 Collection 接口 …… 62

4.5 List 接口 …… 62

4.5.1 List 接口简介 …… 62

4.5.2 ArrayList 集合 …… 63

4.5.3 LinkedList 集合 …… 64

4.6 Set 接口 …… 64

4.6.1 Set 接口简介 …… 64

4.6.2 HashSet 集合 …… 65

4.6.3 TreeSet 集合 …… 65

4.7 Map 接口 …… 65

4.7.1 Map 接口简介 …… 65

4.7.2 HashMap 集合 …… 66

4.7.3 TreeMap 集合 …… 67

4.8 常用的三个工具:Iterator 接口、Collections 类、Arrays 类 …… 68

4.8.1 Iterator 接口 …… 68

4.8.2 Collections 类 …… 69

4.8.3 数组工具类 Arrays …… 70

4.9 图书管理系统 V4.0 …… 71

4.9.1 运行效果图 …… 71

4.9.2 类结构示意图 …… 71

4.9.3 代码实现 …… 71

第 5 章 数据存储 …… 74

5.1 本章任务 …… 74

5.2 IO …… 74

5.2.1 基本 IO …… 74

5.2.2 更好用的 IO …… 75

5.3 文件系统 …… 79

5.3.1 按字节读取 …… 79

5.3.2 按字符读取 …… 80

5.3.3 按行读取 …… 81

5.3.4 随机读取 …… 81

5.4 图书管理系统 V5.1 …… 83

5.4.1 运行效果图 …… 83

5.4.2 类结构示意图 …… 83

5.4.3 代码实现 …… 83

5.5 数据库 …… 89

5.5.1 JDBC 简介 …… 89

5.5.2 JDBC 访问数据库的基本过程 …… 89
5.5.3 JDBC 常用类 …… 93
5.6 MVC 设计模式 …… 95
5.6.1 什么是 MVC 设计模式 …… 95
5.6.2 为什么要使用 MVC 设计模式 …… 95
5.7 图书管理系统 V5.2 …… 95
5.7.1 运行效果图 …… 96
5.7.2 类结构示意图 …… 96
5.7.3 代码实现 …… 96

第 6 章 界面 …… 105

6.1 本章任务 …… 105
6.2 画画的故事 …… 105
6.3 容器 …… 105
6.3.1 底层容器 …… 106
6.3.2 面板容器 …… 108
6.4 布局管理器 …… 108
6.4.1 布局管理器概述 …… 108
6.4.2 FlowLayout 布局管理器 …… 109
6.4.3 BorderLayout 布局管理器 …… 109
6.4.4 GridLayout 布局管理器 …… 110
6.5 组件 …… 110
6.5.1 组件概述 …… 110
6.5.2 常用的组件 …… 110
6.6 事件监听器和内部类 …… 111
6.6.1 事件处理模式 …… 111
6.6.2 事件类 …… 112
6.6.3 内部类 …… 113
6.7 图书管理系统 V6.0 …… 114
6.7.1 运行效果图 …… 114
6.7.2 类结构示意图 …… 115
6.7.3 代码实现 …… 115
6.8 把我们的界面变漂亮 …… 129

第 7 章 网络编程 …… 131

7.1 本章任务 …… 131
7.2 网络的几个重要问题 …… 131
7.3 TCP 编程 …… 134

7.3.1 什么是 TCP/IP …… 134
7.3.2 TCP 建立连接步骤(阻塞式) …… 135
7.3.3 TCP 建立连接步骤(非阻塞式) …… 135
7.4 UDP 编程 …… 136
7.4.1 什么是 UDP …… 136
7.4.2 UDP 建立连接步骤 …… 137
7.5 HTTP 编程 …… 137
7.5.1 什么是 HTTP,为什么要有 HTTP …… 137
7.5.2 HTTP 建立连接步骤 …… 138
7.6 客户/服务器模式 …… 138
7.6.1 控制台上的简单输入输出 …… 139
7.6.2 控制台上的循环输入输出 …… 139
7.6.3 一个客户端和一个服务器一次通信 …… 140
7.6.4 一个客户端和一个服务器多次通信 …… 144
7.6.5 多个客户端和一个服务器串行通信 …… 149
7.6.6 多个客户端和一个服务器并行通信 …… 152
7.6.7 客户端与服务器端 HTTP 通信 …… 155
7.7 图书管理系统 V7.0 …… 158
7.7.1 运行效果图 …… 158
7.7.2 类结构示意图 …… 158
7.7.3 通信协议 …… 160
7.7.4 关键代码 …… 161

第 8 章 多线程 …… 178

8.1 本章任务 …… 178
8.2 几个概念 …… 178
8.2.1 进程 …… 178
8.2.2 线程 …… 179
8.3 生命周期 …… 180
8.3.1 线程生命周期概述 …… 180
8.3.2 为什么要有生命周期 …… 180
8.3.3 线程生命周期详解 …… 181
8.4 线程调度和线程优先级 …… 182
8.4.1 线程的调度 …… 182
8.4.2 线程的优先级 …… 182
8.5 创建线程的两种方式 …… 183
8.5.1 继承 Thread 类创建线程类 …… 183
8.5.2 通过 Runnable 接口创建线程类 …… 183

8.6 线程常用方法 …… 184
8.7 线程同步 …… 184
8.7.1 线程同步理解 …… 184
8.7.2 线程同步实现 …… 185
8.8 计时器 Timer …… 185
8.9 图书管理系统 V8.0 …… 187
8.9.1 运行效果图 …… 187
8.9.2 类结构示意图 …… 187
8.9.3 代码实现 …… 188

第 9 章 扩展——从图书管理系统到 QQ …… 191

9.1 本章任务 …… 191
9.2 总体结构 …… 191
9.3 服务器端 …… 192
9.3.1 运行效果图 …… 193
9.3.2 类结构示意图 …… 193
9.3.3 代码实现 …… 194
9.4 客户端 …… 198
9.4.1 运行效果图 …… 199
9.4.2 类结构示意图 …… 200
9.4.3 代码实现 …… 201

第 1 章

Java 入门

作为本书的开始，强烈建议读者阅读前言，那里有关于编写这本教材的想法和初衷，可以更好地帮助读者理解和学习本教材。

1.1 本章任务

先说明一下，本书每一章的第一节都有名为“本章任务”一节，这一节又分为两部分：一部分为理论任务，就是这一章要学习的几个原理性知识点；另一部分为实践任务，就是围绕理论学习任务，以全书示范案例《图书管理系统》为贯穿，在当前章节理论学习任务下的实践性任务。本章最简单，具体如下。

理论任务：学习一些 Java 的入门知识，这些知识，本章各节目录的标题基本可以概括了。

实践任务：分别在控制台、Eclipse 下开发第一个只有一行输出“欢迎进入图书管理系统”，实现的样子如图 1-1 所示。



图 1-1 本章任务截屏

1.2 Java 的故事

Java 是一门高级计算机程序设计语言。所谓高级，是相对汇编语言那样的低级程序设计语言来说，高级语言对于程序员来说使用更简单、编程更容易。没学过汇编语言的同学看到这里完全不用担心，本书的内容和汇编语言没有关系。

Java“刚出道”的时候不叫 Java，叫 oak，是由 SUN 公司推出的，显然，oak 这个名字没有取好，从 1991 年出道，一直不温不火。后来，SUN 公司又号称为 oak 赋予更多的特性，并在 1995 年命名为 Java，这一改，火了！

当然，每个火了的语言一定有其自身的努力和优势，对 Java 来说，有以下的优势。

- 简单。Java 出现前使用普通的三种语言 C/C++/VC，相比这几门语言，Java 是晚

辈，晚辈的好处就是把前辈烦琐的、语法中有的但实际没什么意义的东西都摒弃了，例如指针、多重继承等。学过微软 VC 的同学都知道用 MFC 写个 helloworld 都需莫名其妙地生成一大堆 framework 的代码，而这些在 Java 里就不存在了。

- 跨平台。说到跨平台，先做个比喻，大家都知道世界上有很多种语言，在人类语言里，Java 就好像是国际语，国际语有国际语的好处，就是到哪都讲得通，都能运行，在计算机世界里，我的 64 位机上的. exe 文件放在你的 32 位机上可能就不能运行，但 Java 程序编译后生成的不是. exe 文件，而是. class 文件，如图 1-2 所示。这里我想到一道以前的面试题，大致意思是问“一个 int 型的变量在 C 和 Java 中各占多少个字节”，这道题答案是：在 Java 中 int 总是 4 个字节，但 C 中根据不同的平台所占字节是不一样的，在 16 位 C 编译器中，int 是 2 字节，在 32 位 C 编译器中 int 为 4 字节，在 64 位 C 编译器中 int 为 8 字节。那么 Java 的这种跨平台有什么好处呢？例如一个超级富豪在银行的存款本来是放在一个 64 位计算机上，有一天这台 64 位机出了问题，这个存款被备份到另一台 16 位计算机上，这下惨了！这位富豪的存款不仅全无，而且还负债累累，因为数字太大，放在 16 位计算机上溢出了，变成一个负数了。如果这个程序是用 Java 编写的，就不会有这个问题，因为不管在什么平台上，Java 的 int 变量总是 4 字节。

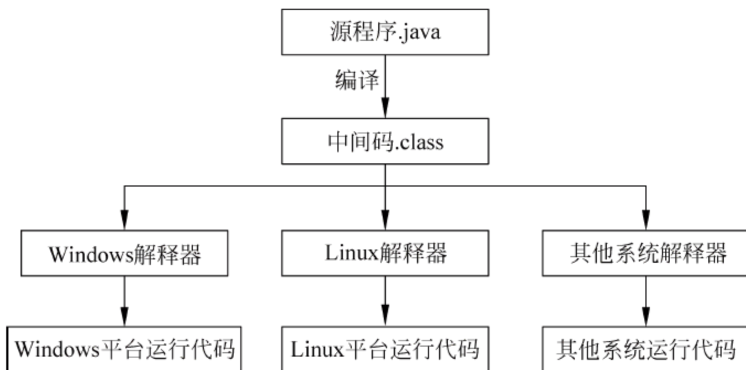


图 1-2 Java 跨平台原理示意图

国际语的另一个特点就是到哪去都要带不同的翻译，各位从图 1-2 中就可以发现这个问题，一个说国际语的人到我们这就要带个会说汉语的翻译，要是去了俄罗斯就得带个会说俄语的翻译。一个不会说国际语的人，例如我只会说中国话，我不能跨平台，但我在咱们自己国家到哪都不用带翻译，说话办事交流方便、快！落实到计算机世界，大家都知道，不管是什么语言什么程序，最终计算机能理解执行的只有 0/1 组合的二进制可执行文件，C/C++语言开发的程序编译后就直接是可执行的. exe 文件了。但 Java 为了跨平台成为国际语，编译后生成一种中间代码. class 文件，再由不同平台的翻译去解释执行. class。根据目标平台(如 Windows、Linux、mac 等)的不同，这些翻译分别叫 JVM for Windows/Linux/Mac…，所以，用 Java 开发的桌面程序，尽管现在的 Java 提出了很多优化改进的方法，用户还是普遍感觉“慢”，这也是 Java 跨平台的一个代价吧。特别是界面部分，Java 为了跨平台，往往用 Java 开发的界面还有点“丑”，没办法，这些都是跨平台的代价。当然，

本教材的后面会教大家把这些界面变漂亮的方法(参见 6.8 节)。不过,漂亮也有代价,就是不能跨平台。

说明:如果你现在还不知道“编辑、编译、运行、调试”,还有 JVM 的意思,不要紧,后面会讲到,正如我在前言部分谈到一个大学学习的特点,我们大学的学习不可能像中学那样学到第 5 页的时候总是可以用前 4 页的知识解释第 5 页,定理不能解释的给你来个公理就 ok 了。对于大学的学习,有些知识点你在第 5 页不懂的时候一定要耐心读下去,也许你在读到第 50 页的时候就明白了。

- 市场需求大。不管如何,市场需求是硬道理,我们同学绝大部分毕业后都希望找份好工作。不过,读到这里的同学还是要有个疑问,就是刚才我们才说过 Java 开发的桌面程序有点“慢”,那么为什么 Java 还这么火? 这个问题从 Java 火的时候到现在就在不停地被讨论,下面也只能谈谈我的看法供同学们参考,要想了解得更多,需要同学们自己“百度”了,并且“百度”后还需要同学们自己领会、融合、参悟。Java 之所以有这么大的市场需求,我想有以下几点:(1)Java 庞大而完善的生态系统。从桌面 Java 到当下手机 Android 到网页 Java Web,再有大量成熟的 Java 开源项目可以在我们遇到问题时方便地复用,具体不多说了。(2)Java 的优势在于手机上的 Android 和基于网页的 Java Web,几乎没有哪个本地桌面应用是用 Java 做的。那么,问题又来了,既然 Java 本身都慢,基于网页的服务器端的 Java Web 和手机上的 Android 就不慢了吗? 我们为什么不直接学习 Java Web 或 Android 呢?

  首先,解释第一个问题可能同学们在这个阶段有点吃力,但是我们还是要知道一点点,就是 Java 更适合开发基于 Web 的服务器端的程序(那什么是 Web 呢? 哎呀,越写越发散了,但是我发现实际教学中很多同学是不懂的,同学们可以理解 Web 程序就是基于网页的、你用浏览器使用的应用程序),相比网络 I/O 读写,Java 在服务器上实在是太快了。还有,Java 的“慢”请同学们一定不要过分解读,Java 的性能已经足够优秀了,已经优秀到在全世界拥有最大数量的程序员了。

以上这些特性是作者认为比较重要的,不同的教科书或网上内容说法会有些出入,例如安全、并发等特性,这不要紧,就像对于一个大家都喜欢的歌手,你喜欢这位歌手的点和我喜欢的点可能不一样,但是那不要紧,这些问题没有标准答案。

另外,关于 Java 的特性,还有“面向对象”这一项我想和同学们交流一下。面向对象,特别是纯面向对象,是 Java 的重要特性,也只能算是一个特性,说是优势有点过了,对于面向过程和面向对象,只是两种不同的设计思想,没有好坏之分。至于什么是面向对象,如何面向对象,在后面的章节和实践代码中我们会慢慢学。在此,用一个简单的例子看看面向对象和面向过程的不同思想,例如“张三是个见人说人话见鬼说鬼话的人”这句话:

用面向对象的思想就是:zhangsan.talk(person)或者是 zhangsan.talk(ghost)。

用面向过程的思想就是:talk(zhangsan, person)或者是 talk(zhangsan, ghost)。

说到 Java 的纯面向对象,其实也是有和 C++ 比较的时代特点,因为在 Java 刚出道的

那些年，市场上流行的主流语言 C++是一种可以把类和函数混搭的语言(至于 C++为什么会出现这种混搭现象、什么是类、什么是函数，如果有同学还不知道，不要紧，耐心地往后学习)。

1.3 三个版本的过去和现在

为了覆盖手机、桌面和网页，Java 在过去有三个版面，分别叫 J2ME(手机)、J2SE(桌面)和 J2EE(网页)，这里“2”的意思是在 Java1.2 版本以后称为 Java 第二代。后来为了统一 Java 版本更新的叫法，就分别叫 Java ME、Java SE 和 Java EE 了。

而当下，除了 Java SE，也就是我们这本教材学习的 Java 外，Java ME 已经被主流的 Android 平台取代，Android 是一个面向 Java 语言由 Google 公司开发的手机操作系统，它的最大特点是开放源代码，所以现在市面上有很多基于 Android 的系统。对于 Java EE，由于目前关于 web 的 Java 实现的轻量级项目像 SSH、Spring 等被众多的企业所使用，所以同学们会在市面上看到很多教材都在讲 Java Web。

有兴趣学习 Android 和 Java Web 的同学可以参阅我的另外两本书《Android 程序设计教程》《Java Web 程序设计教程》。这里还想说一下学习顺序的事情，不管您有没有兴趣学习其他的课程，对于 Java 的三个系统，您首先应该学好 Java 本身，也就是这门课和这本书的知识点。

1.4 环 境 搭 建

学习 Java，我们要安装的软件是很少的，就 JDK 和 Eclipse 两个软件，下面分别对它们进行介绍。

首先，JDK 的下载、安装非常简单，大家在百度里搜“JDK”第一个链接就指向它的官网下载，安装时一直单击“下一步”即可。当然，不会的同学可以百度一下“Java 环境搭建”。这里，我不想把一步一步的截屏写在这本教材里，这点基本的能力作为大学生的我们需要具备。不过，这里我想解释以下 4 个问题。

(1) 如何判断你安装好了?

单击“开始”菜单→“运行”，在弹出对话框中输入“cmd”，如图 1-3 所示。



图 1-3 “运行”对话框

按 Enter 键，进入命令行窗口，如图 1-4 所示。

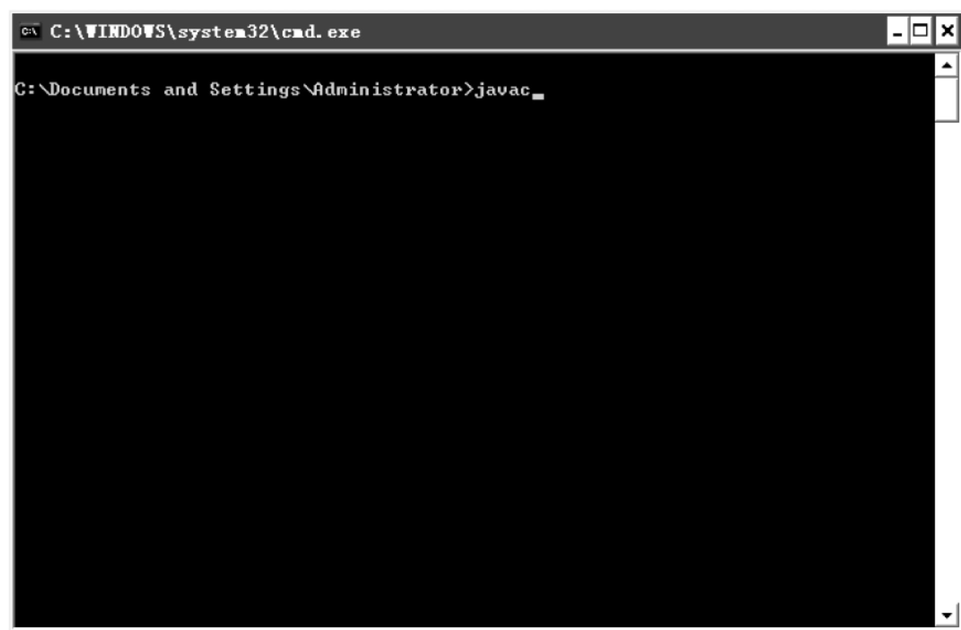


图 1-4　cmd 命令行界面

在行命令下输入"java""javac"，按 Enter 键，如果弹出类似如图 1-5 中一大堆的提示，就说明安装成功了。

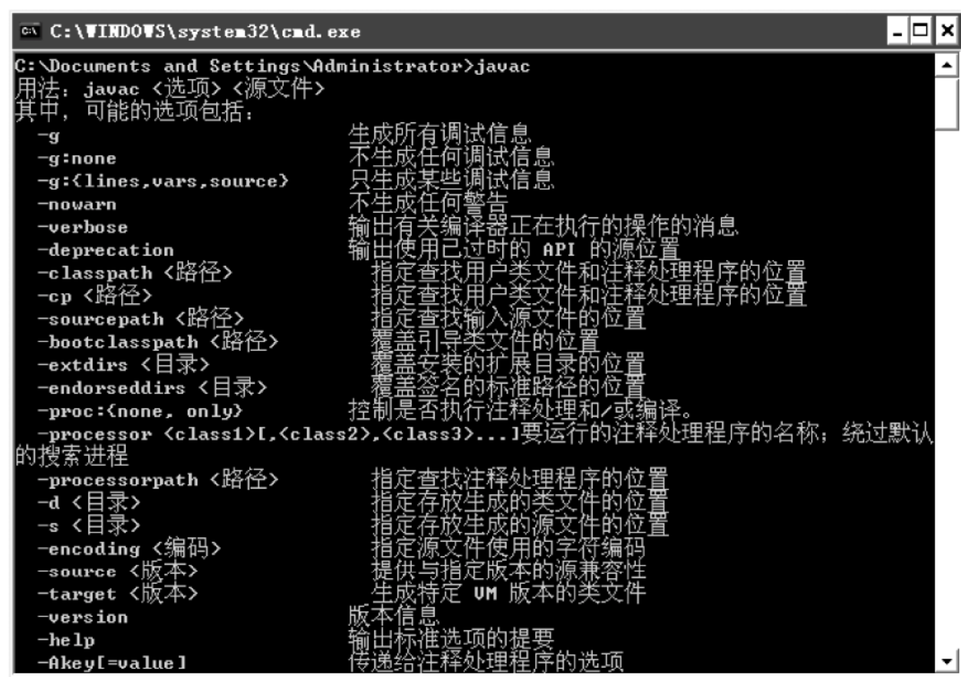


图 1-5　cmd 下输入 javac 后的界面

(2) 为什么要在安装完 JDK 后配置 path 和 classpath 两个参数(这里假设大家使用的是 Windows)?

这里要说明下为什么要配置这两个参数,配置 path 的目的是告诉操作系统刚才输入的 java、javac 等命令在什么路径下,因为这些命令不是操作系统自带的。

配置 classpath 的目的是我们的程序会不停地调用 java 提供的类库,例如最简单的 System.out.println(…)函数也是 java 提供的,我们在使用 javac 编译程序的时候,这里配置 classpath 就是告诉操作系统去哪里找到 java 提供的类库。不过,如果同学使用的话可以不用配置 classpath,因为 IDE 自己可以找到 java 的安装路径。初学者在配置环境变量 classpath 时有一点要注意,就是在原有的 classpath 最前面加上".;"符号,这是表示在原有路径配置下追加的意思。

(3) Eclipse 是什么?

开发一个程序,我们通常要有以下几步:编辑→编译→运行→调试→再运行→再调试→……→发布,这个过程我们可以在 cmd 下用 javac、java 等行命令来操作,但是对于大一点的程序或者即便是一个普通的小项目,都实在太麻烦了。

(4) 关于软件版本

初学的同学对于软件版本经常怀疑装错了。就本书学习的知识,同学们装哪个版本其实都不重要,我在写书的时候 JDK 是 8.0 的版本,其实同学们在网上几乎是随便找个版本就够了,因为这本书涉及的知识点都是最基本的。

1.5 从控制台到 Eclipse——一个最简单的图书管理系统 V1.0

从这一节开始,我们就进入第一个程序,同时,我们的"图书管理系统"也已经上线了。作为第一个程序,这里还是手把手带着大家做一次。

第一步:打开 Eclipse,如图 1-6 所示。

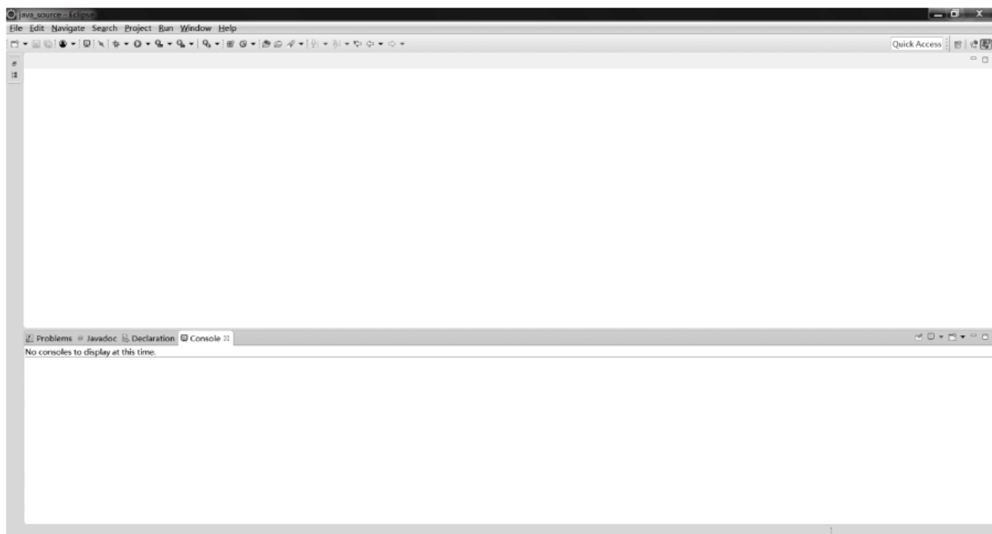


图 1-6 Eclipse 主界面

第二步：单击 File→"新建"→Java Project，给我们的第一个 Java 项目取个名字，就叫 Chapter1.5，单击 Finish 按钮，如图 1-7 所示。

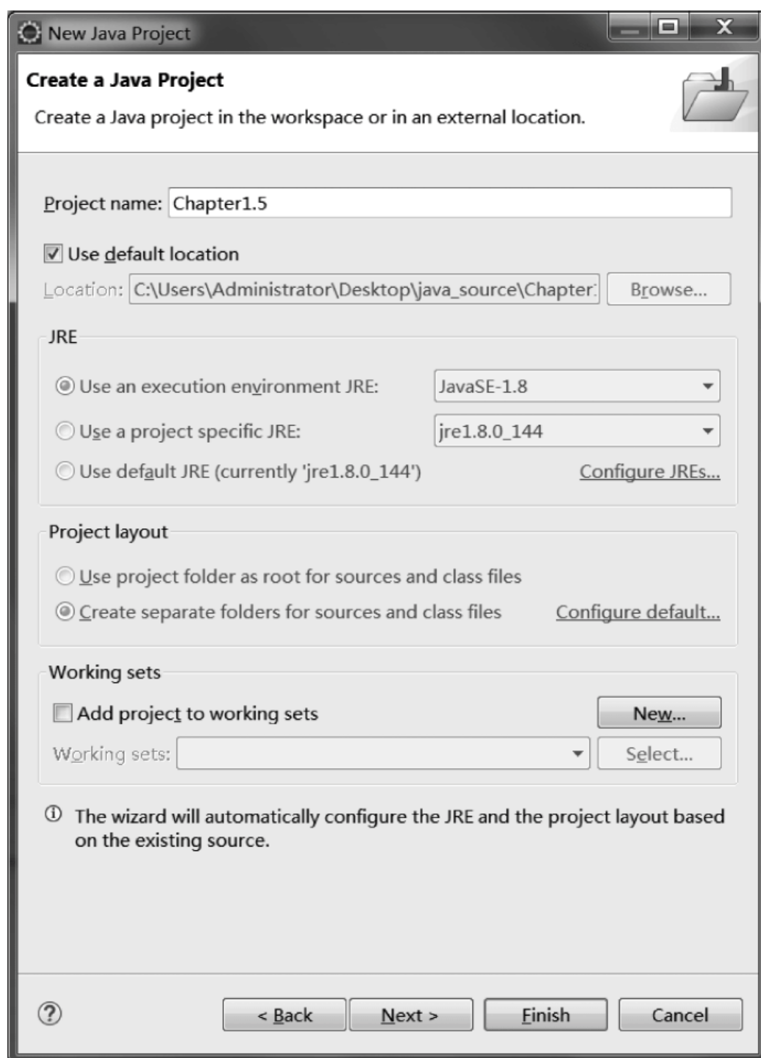


图 1-7　创建 Chapter1.5 截图

第三步：右击这个项目，选择新建包 ui，如图 1-8 所示，图 1-9 为创建后项目结构的截图。

这里，顺便说明下什么是"包"——我们知道在使用电脑时，每个不同的程序或者文件都会被系统自动或者人为地分配到某个文件夹中，为什么需要这样的操作呢？原因在于，通过检索这些不同的文件夹，我们便能够快速地找出自己所需要的文件，并且使用文件夹能够进一步地帮助我们完成文件的归类存储。同样地，在 Java 程序中，我们可能会创建许许多多的文件，这些文件有着不同的功能和特征，我们将其放入不同的文件夹，这些文件夹就称为包(package)。包就像是一个目录结构，通过创建包，我们可以在使用或者查找包中文件时提高效率。

第四步：右击 ui 包，新建一个类 MainClass，勾选 public static void main(String[]args)选

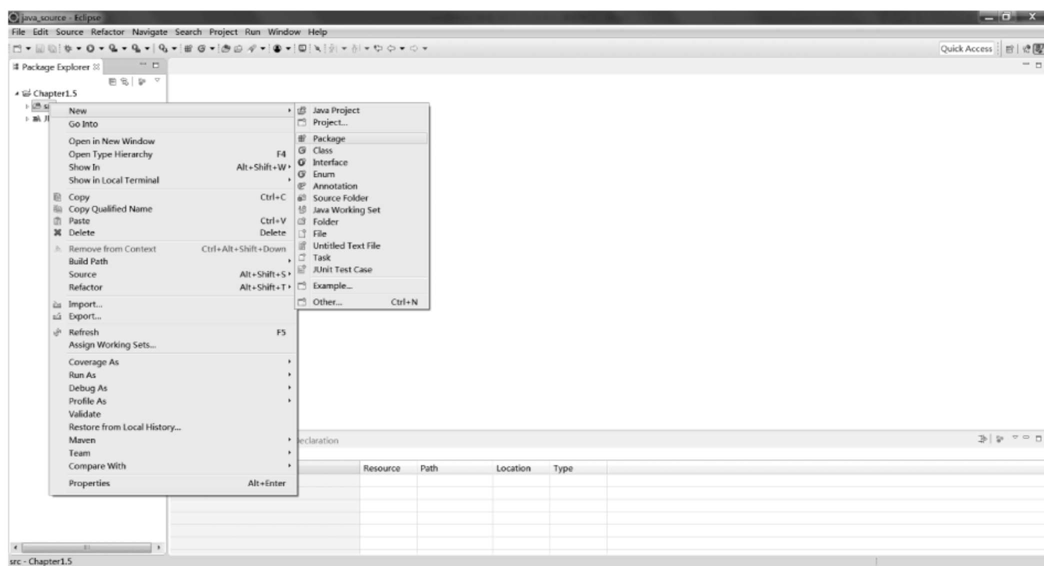


图 1-8 创建 ui 包截图

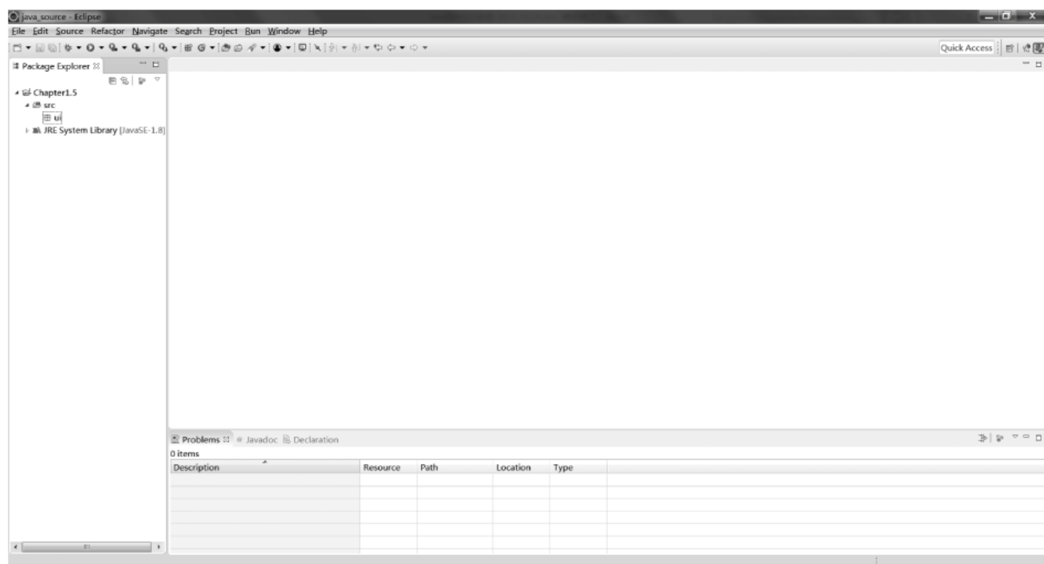


图 1-9 创建后项目结构截图

项,在 main 函数中写一行代码：System.out.println("欢迎来到图书管理系统!")；整体工程如图 1-10 所示。

第五步：我们完成代码后,在代码空白处右击出现菜单,选择 Run as→Java Application 后,我们编写的程序即可运行,如图 1-11 所示。

其中,类 MainClass 的代码如下：

MainClass.java

```
package ui;

```

```
public class MainClass {
public static void main(String[] args) {
            System.out.println("欢迎来到图书管理系统!");
        }
}
```

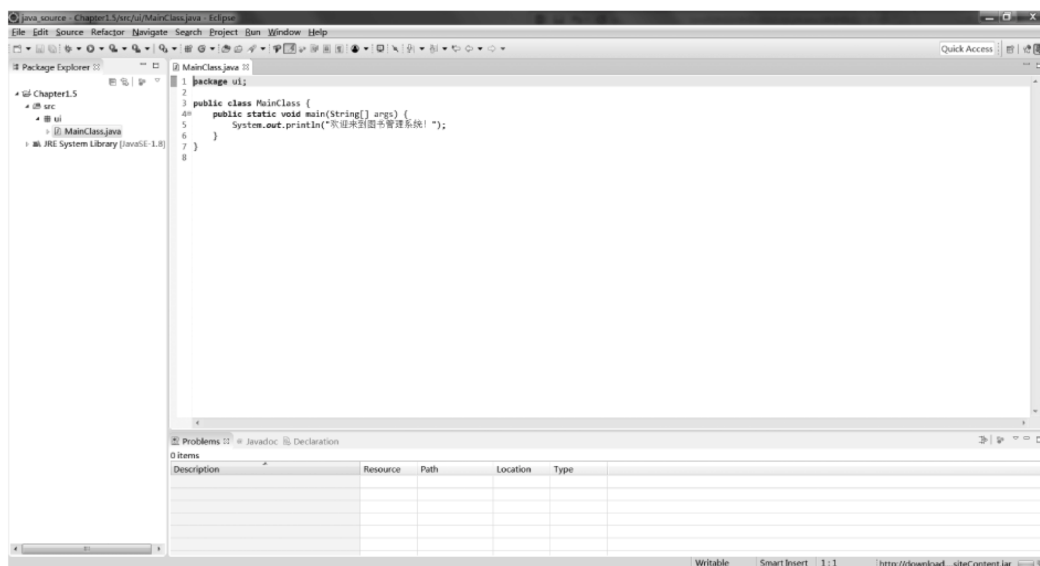


图 1-10　项目总体结构截图

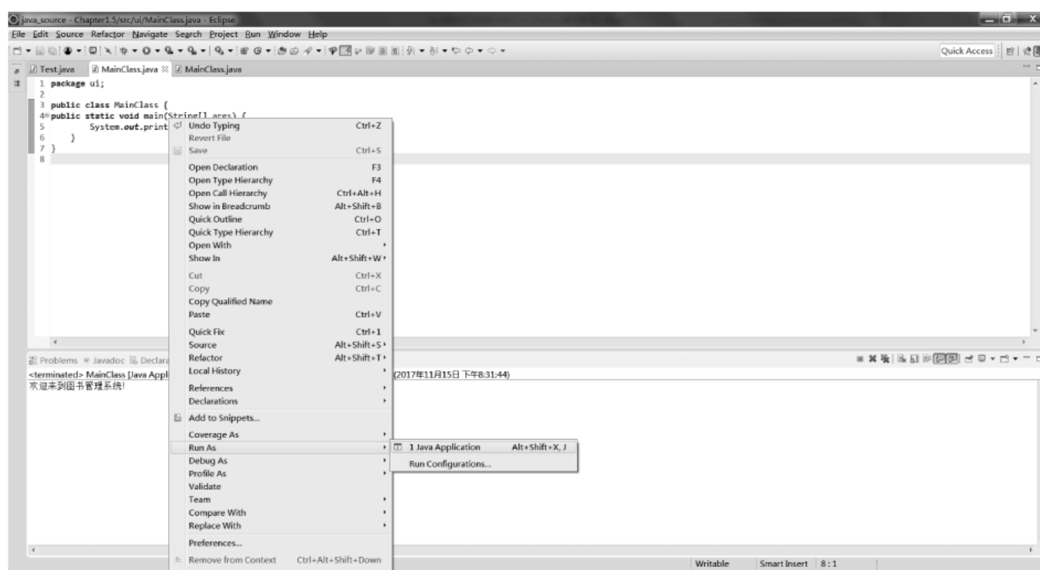


图 1-11　运行代码操作截图

至此，我们的第一个 Java 小程序就完成了，我们自己的图书管理系统也就“开张”了。

1.6 一个简单的面向对象的改造

在1.5节中，我们看到了图书管理系统的雏形，我们在Java程序的入口函数——Main函数中，编写了输出欢迎词的打印语句。这样做的确很方便，编写整个程序的逻辑之后，程序就可以直接编译执行了。看上去很完美，不用过多考虑其他的代码规划或者概念。在Main函数中编写我们的程序，这种思想就是面向过程的编程思想，当把代码放入对应的类中去后，我们的编程思想就变为面向对象了。

如果拥有了对象这一个概念之后，对于每个我们所需要的模块和功能都能进行单独的使用，这样代码量会减少，模块划分会更为清晰，程序可读性也会随之上升。为了切实学习到Java语言最为基础也是最为精华的编程理念，我们从new出第一个对象开始，图1-12展示了对象改造后的系统运行截图。在后边的第三章会对对象这一概念进行详细的讲解。

MainClass.java

```
package ui;

public class MainClass {
    public MainClass() {
        System.out.println("欢迎来到图书管理系统!");
    }

    public static void main(String[] args) {
        new MainClass();
    }

}
```

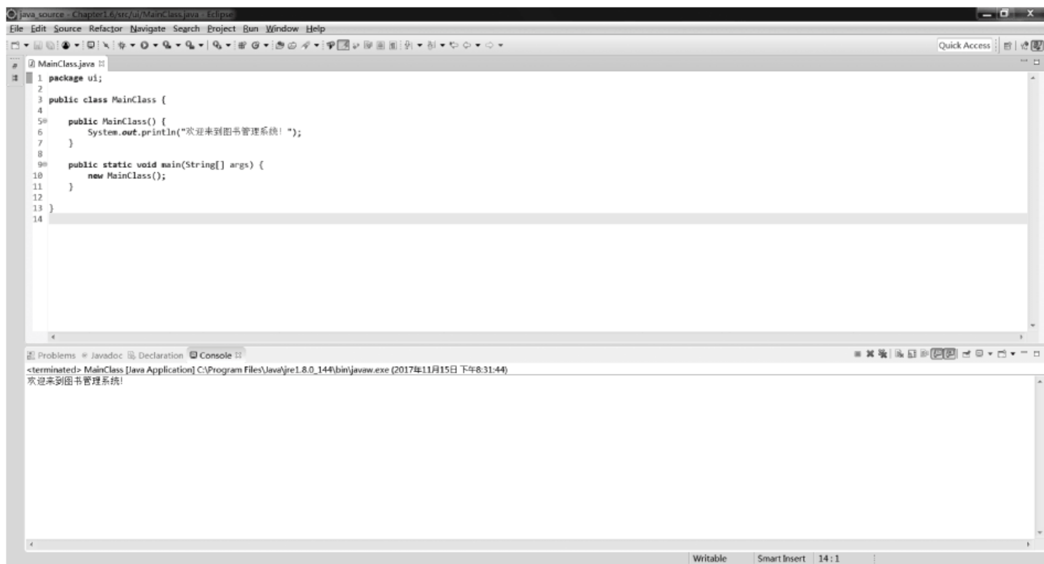


图1-12 进行对象改造后的系统运行截图

改造后的图书管理系统是通过 new 出一个 MainClass()对象来对相关的代码进行调用的。是不是很简单呢？当然，对于一个系统而言，只有一个欢迎表语可是远远不够的，我们将在第 2 章的学习中，为图书管理系统添加一些使用的功能。在第 2 章之前，我想让同学们对于 Java 编程语言中的几个基本的概念有所了解。

1.7　JDK、JRE 和 JVM

在第 1 章的开头，我们已经在自己的电脑中，安装了 Java 的运行环境、配置了环境变量。在配置环境变量时，细心的同学就会发现在环境变量的配置中，Java_HOME 这一变量需要配置 Java 安装路径。而在安装路径中，有两个文件夹，第一个是 jdk 开头的，第二个是 jre 开头的。这时候我们需要选择 jdk 开头的文件夹作为路径使用，而不使用 jre 文件夹的路径。而且，我们为什么要配置这些环境变量呢？我们的 Java 程序又是怎样执行的呢？大多数初学 Java 的同学都会被这些庞杂的概念给搞迷糊，那么，笔者现在就为大家解释一下这几个概念。

首先是 JDK，JDK 的全称为 Java Development Kit，译名为 Java 开发工具包。Java 开发工具包是 Java 环境的核心组件，并提供编译、调试和运行一个 Java 程序所需的所有工具，可执行文件和二进制文件。JDK 是一个平台特定的软件，有针对 Windows、Mac 和 Unix 系统的不同的安装包。JDK 就好像工具箱中的工具，要使用工具，我们就必须有这个工具箱——JDK。

其次是 JRE，JRE 的全称为 Java Environment Runtime，译名为 Java 运行时环境。JRE 是 JVM 的实施实现，它提供了运行 Java 程序的平台。JRE 包含了 JVM、Java 二进制文件和其他成功执行程序的类文件。JRE 就像是一个处理工具，通过 JRE 即可运行 Java 开发的应用程序。但是，它却没有包含有 JDK 中的开发工具，只是支持程序的基本运行，而不支持程序的编写和开发。

最后是 JVM，JVM 的全称是 Java Virtual Machine，译名为 Java 虚拟机。JVM 实现了 Java 至关重要的特性——跨平台。Java 开发的程序代码，经过编译后，程序并不在电脑的 CPU 上运行，而是将其交由 JVM 代替执行。这也是 Java 语言的灵魂所在，整个 Java 异彩纷呈的程序王国也都是在 JVM 之上演绎。

以上分别介绍了 JDK、JVM、JRE 三者的含义，现在我们来总结三者的区别。

(1) JDK 是用于开发的，而 JRE 是用于运行 Java 程序的。

(2) JDK 和 JRE 都包含了 JVM，从而使得我们可以运行 Java 程序。

(3) JVM 是 Java 编程语言的核心并且具有平台独立性。

相信仔细阅读本章节的内容，我们也对这些专业术语有了基本的了解，通过后期的学习和积累，对这些概念的理解也会愈加深入。

1.8 养成良好的学习习惯

在前言部分已经讲过，我在课堂上常要求同学们大学期间在专业课学习上做到三点：(1)扎实的专业基础知识；(2)良好的英文读写水平；(3)快速掌握陌生知识的能力。本书主要作为本科教材，因此这不是一本很厚的、面面俱到的 Java 书，而且我们认为本科的教学本身也应该是启发式的教学。对于 Java 的学习，也是一样，一定要掌握原理，理论性知识和上机实践同样重要。

到目前为止，同学们要用好以下几种资源：(1)书；(2)Eclipse；(3)API 文档，也就是 Java 的帮助文档；(4)网络：用好百度、谷歌。

1.9 如何导入本书的案例库

本书所有涉及的源代码，我们均会在网盘中保存，如果同学们需要，可以扫码下载。在下载后，我们如何将源码在编译器中打开呢？

第一步，我们先打开 Eclipse，单击 File 选项中的 import，如图 1-13 所示；

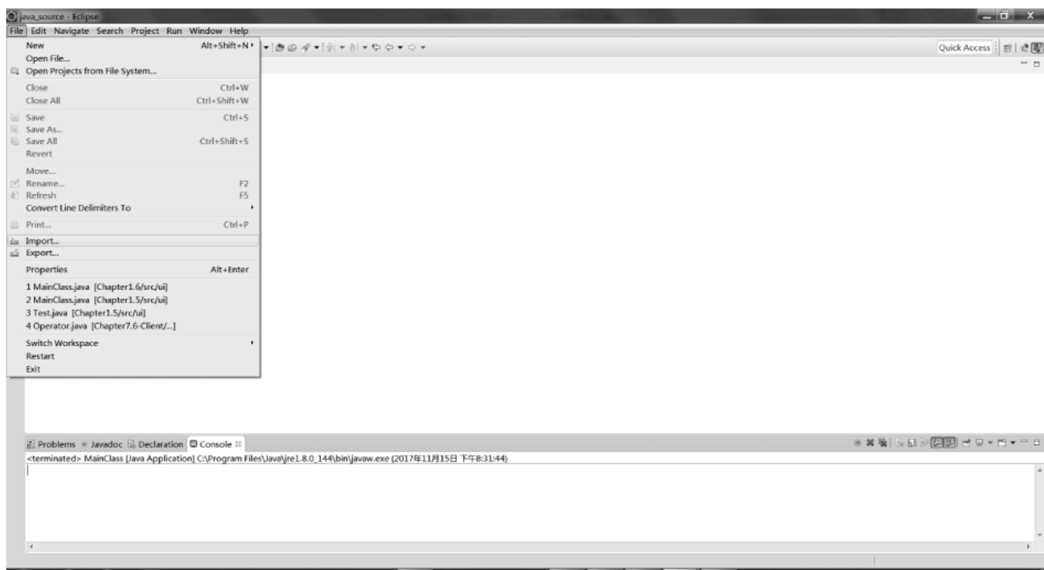


图 1-13 import 功能选项截图

第二步，进入 import 界面后，我们选择 Existing Project into Workspace，如图 1-14 所示；

第三步，选择 Browse，选择你下载好的源码所在的存储位置。假设我的存放位置为"C:\Users\Administrator\Desktop\java_source\Chapter1.5"。选择好文件后，单击 Finish 按钮，项目则会自动导入到工作空间中，如图 1-15 所示。

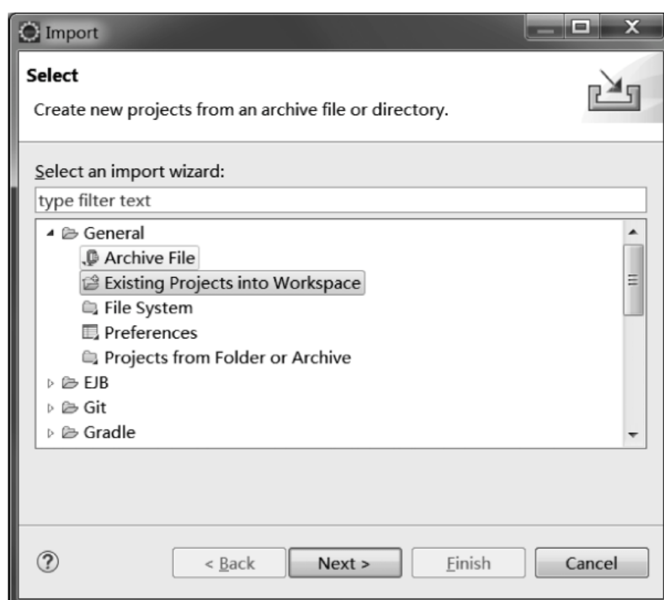


图 1-14　import 选项截图

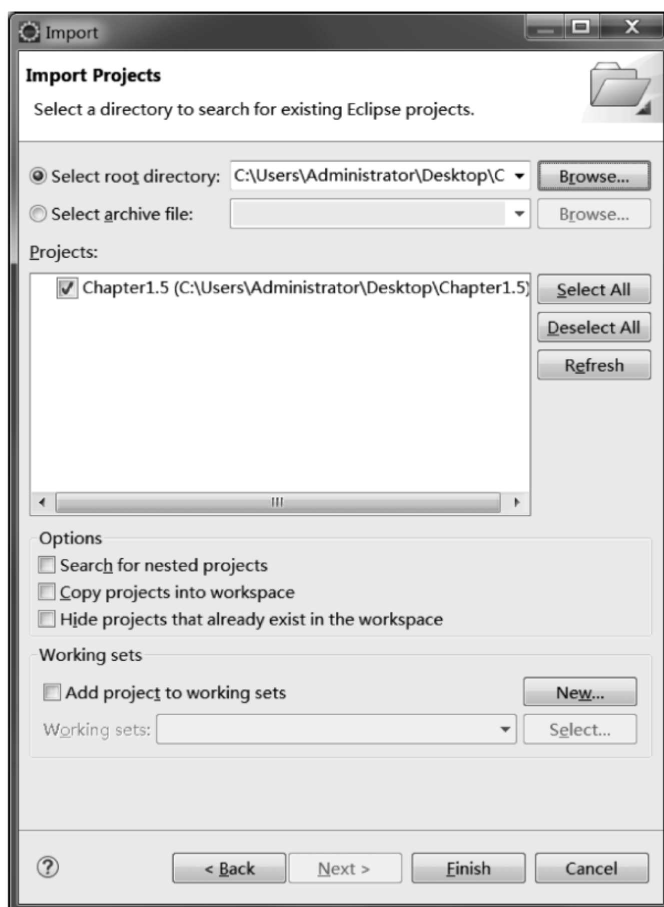


图 1-15　导入项目选项截图

第 2 章

基础知识

在第 1 章中，我们配置了 Java 的环境，简单地编写了 Java 的代码，在本章中，我们会细致地讲解 Java 的基础知识。

2.1 本章任务

理论任务：学习 Java 程序的基本结构、类、包、变量、控制语句的基本概念和使用、常识性代码写作规范。

实践任务：实现一个没有界面、没有数据库、没有网络的本机版的黑底白字的、控制台的图书管理系统。具体地，有一个简单的主菜单，上面有“增加图书请选 1；删除图书请选 2……退出系统请选 0”，在选择进入任何一个具体功能后有具体的用户简单操作。图书管理系统实现后的截图如图 2-1 所示。

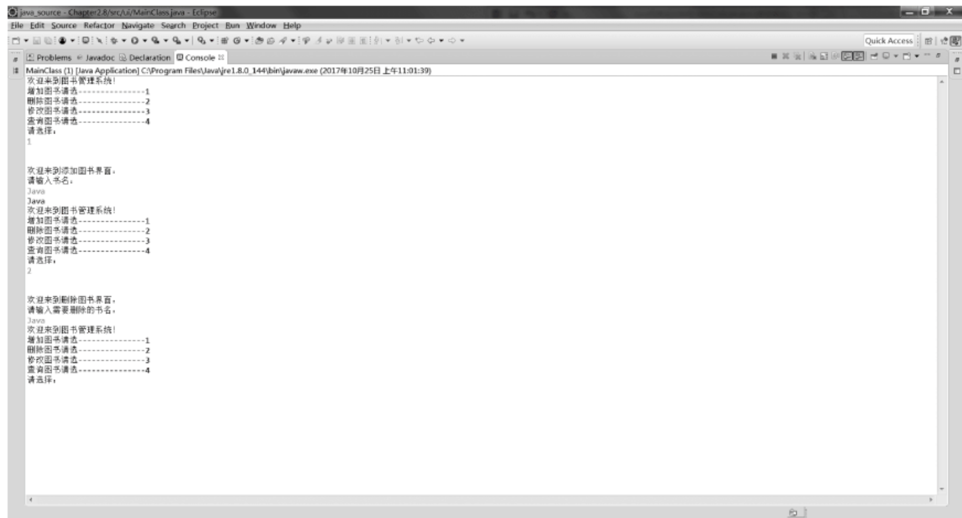


图 2-1　图书管理系统运行效果截图

2.2 Java 基本程序结构

在第 1 章的学习过程中，我们对于 Java 语言有了一个大体的认识：Java 的过去、如何安装运行 Java 程序、Java 程序的优势。在第 1 章中，我们的第一个图书管理系统开

张了。通过编写打印语句和 new 对象，我们学会了两种在控制台上输出“欢迎来到图书管理系统”这句话的方式。但就这短短的 10 行代码，同学们你们能够说出每个单词在 Java 语境下的具体含义吗？他们的具体用途是什么你们有所了解吗？我想你们现在肯定也是一头雾水，没关系，那么我们就开始讲述一下关于 Java 程序结构的一些基本知识点。

Java 是一门面向对象的编程语言，所以在 Java 中一切都是对象，数据和函数都必须封装在“类”中。在第 1 章的代码中，我们定义了一个类(MainClass)。首先关键字 public 称为访问修饰符，用于控制其他代码对被修饰类的访问权限。class 是声明类必须的关键字，这也体现了 Java 程序中所有内容都包含在类中的特点。类名(MainClass)紧随关键字 class，类名通常以大写字母开头，后面可以数字字母任意组合。如果类名称由多个单词组成，则每个单词的首字母均应为大写(驼峰命名法)，类名 MainClass 就遵守了这样的规范。必须谨记的是不能使用 Java 保留字(例如：public、private、class 等)来命名。在命名一个资源时候，应该本着描述性以及唯一性这两大特征来命名，才能保证资源之间不冲突，并且每一个都便于记忆。虽然这些命名法则并没有在 Java 编程中有强制的要求。但好的命名方式可使项目的代码样式统一，具有良好的可读性。

正如我们在第 1 章中所提及的，Java 中有一个包的概念。一个包就是一个类库单元，包内包含有一组类，它们在单一的名称空间之下被组织在了一起。这个名称空间就是包名。可以使用 import 关键字来导入一个包。当你想使用某个其他包中的函数或者类时，使用“import＋空格＋包名”的形式来引进对应的包。当然，在多数情况下，编译器都可以很智能地为你把使用的函数或类的对应包名自动引入。

当然，Java 区分大小写，如果出现拼写错误就会导致程序无法正常运行(例如：将 main 写成了 Main)。main()是一个入口方法，并且对它有些特殊规定，例如其名称必须为 main()，必须是公有静态方法，有命令行参数等。在 main 方法中，使用了 System. out 对象并调用了它的 println 方法。注意：“. ”号用于调用方法成员变量。调用方法：object. method()。这就是我们使用的打印语句，调用方法的流程，相信通过不断的训练和模仿，你很快就能掌握这几个简单的知识点。

2. 3　Java 程序基本代码规范

在第 1 章的图书管理系统里，我们已经掌握了最为基本的 Java 程序结构，但是，对于整个 Java 语言编写的规范和理解仍然相当于零。所以，在本章接下来的学习中，我们将详细地了解一系列的规范、Java 各个关键字的使用和对它们的理解。一份好的代码，不仅仅是运行时展现的良好性能，更重要的是，它是否能像一篇优美的文章一样，让别人在阅读时，赏心悦目。在未来维护或更新时，为自己也为别人带来便捷。这是我们需要学习的编写代码的态度，更是我们终身受用的优良程序员品质。

2.3.1 注释规范

代码要是没有注释，对读者来说就是一堆乱七八糟的字母，为了提高代码的可读性和可维护性，必须对代码进行必要的注释。注释可以在程序员阅读代码时提供帮助，同时编译器会忽略注释，所以不会影响代码的编译。良好的注释习惯是一个优秀的开发人员需要具备的基本素养。适当的注释能够起到带领读者深入理解代码的作用，但过多则会使得整份代码显得冗长。Java 的注释分为两种：实现注释(implementation comments)和文档注释(document comments)。其中实现注释又分为单行注释和多行注释。

1. 实现注释

(1) 单行注释：当需要对某一行代码进行注释时，在代码后面加上“//”和注释内容。

例如：

```
System.out.println("This is the first I wrote!");        //控制台输出
```

(2) 多行注释：当需要注释多行内容时，可以将注释内容放在符号“/ * ”与“ * /”之间。

例如：

```
/*
 * System.out.println("This is the first I wrote!");
 * System.out.println("I love programming!");
 **/
```

2. 文档注释

文档注释可以用于生成文档。它描述了 Java 的类、接口、构造器，方法，以及字段(field)。就像我们在 1.8 节中提到的 Java API 文档形式那样。每个文档注释都会被置于注释定界符“/ ** ”和“ * /”之中，一个注释对应一个类、接口或成员，该注释应位于声明之前。如果我们使用了文档注释，那么，文档注释中所提及的内容也就会通过自动的编译器生成为对该类或该方法的解释。当我们悬停鼠标在某个具体的方法上时，出现的提示窗中对该方法的解释就是由文档注释生成的。而“@param”标签代表的是该方法中所使用的参数描述，“@return”代表的是返回值的描述，它们也会自动生成为 HTML 格式的文档，形成超链接可单击进行跳转。例如：

示例代码

```
/**
*此函数用于将 String类型的数字转换成整型的数字
*@author Administrator
*@param number String型的数字
*@return int转化后的结果
*/
public static int convertIntoInt(String number) {
  return Integer.parseInt(number);
}
```

通过这样的形式，使用文档注释即可在编译器中有对应的函数使用描述出现。具体效果如图 2-2 所示。

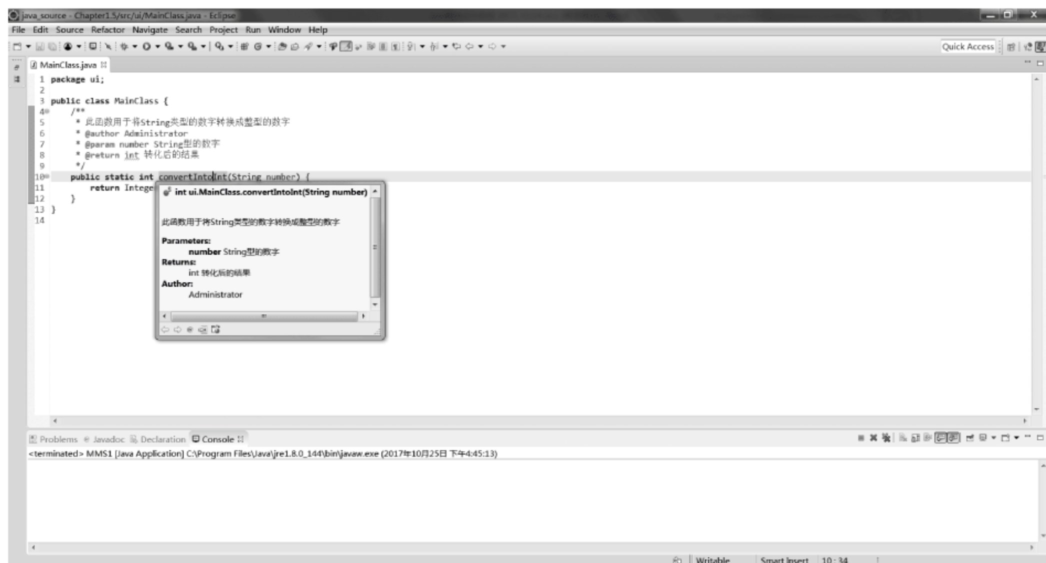


图 2-2　文档注释效果截图

2.3.2　标识符

在 Java 中，变量、常量、函数、语句块等都有名字，我们统统称之为 Java 标识符。标识符是用来给类、对象、方法、变量、接口和自定义数据类型命名的。Java 标识符由数字，字母和下画线(_)，美元符号($)或人民币符号(￥)组成。在 Java 中是区分大小写的，而且还要求首位不能是数字。最重要的是，Java 关键字不能当作 Java 标识符。下面举两个例子来帮助大家区分合法标识符和非法标识符。

合法标识符：myCar、my_Car、Phone、$Phone、_House

非法标识符：#Cat、22Dog、public、class

2.3.3　关键字

1. 关键字

关键词是编程语言里具有特殊含义的词。程序员若想和 Java 平台进行沟通，就得有相互都明白的约定，而关键词就是确保我们和 java 平台进行信息交换的约定。使用关键字就像输入了指令或一个提示词，使编译器和解释器理解这段代码所想要指定的范围或功能。Java 关键字如表 2-1 所示。

表 2-1　Java 所有关键字列表

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else

续表

enum	extends	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	strictfp	short	static	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

在以上列举的关键字中，每一个关键字都有独特的含义及作用。例如：public 关键字是可以应用于类、方法或字段（在类中声明的变量）的访问控制修饰符；new 关键字用于创建类的新实例；import 关键字使一个包中的一个或所有类在当前 Java 源文件中可见。

对于关键字而言，我们在 Java 编程时，有些关键字不能作为变量名进行使用，否则，编译无法通过。我们也不推荐读者使用与关键字相同的方式去命名变量。这样对于阅读和理解代码没有正面积极的帮助。

注意：

(1) Java 中所有的关键字都是小写。

(2) 标识符命名时不能用关键字。

(3) Java 的关键字也是随新的版本发布在不断变动，不是一成不变的。

介绍了这么多与关键字的相关概念和需要我们在使用时注意的事项后，我们就来具体地了解一下 Java 中不同的关键字和它们对应的功能吧。首先是有关访问的修饰符关键字，具体的关键字和其对应的意义如表 2-2 所示。

表 2-2　访问修饰符的关键字

关键字	含义	备　注
public	公有的	可跨包。 用法：public＋具体语法
protected	受保护的	当前包内可用。 用法：protected＋具体语法
private	私有的	当前类可用。 用法：private＋具体语法

第二是与类有关的关键字介绍，与类相关的关键字主要控制你所需要定义的类的具体属性，是否继承了其他的类，是否实现了某个接口。需要注意的是，Java 语言中，只能继承一个父类，但可以实现多个接口。Java 也是通过实现接口从而展现多态这一特点的。与类相关的具体关键字和其对应的意义如表 2-3 所示。

表 2-3　定义类、接口、抽象类和实现接口、继承类的关键字、实例化对象

关键字	含义	备　注
class	类	花括号里有已实现方法体，类名需要与文件名相同 用法：public class A(){}
interface	接口	花括号里有方法体，但没有实现，方法体句子后面是英文分号“;”结尾 用法：public interface B(){}
abstract	声明抽象	介于类与接口中间，可以有也可以没有已经实现的方法体 用法：public abstract class C(){}
implements	实现	用于类或接口实现接口 用法：public class A implements D(){}
extends	继承	用于类继承类 用法：public class A extends D(){}
new	创建新对象	用法：A a＝new A();

第三是与包相关的关键字。在第 1 章的内容中，我们已经介绍过包的概念，使用包可以方便我们快速地查找编写的代码，想要编译器也知晓你现在编写的类和代码在哪个包中，我们就需要使用 package 关键字来告知编译器。同时，如果我们想要引用其他包中类的方法，我们则需要使用 import 关键字，这样才能使编译器查找到相应的类和函数。具体与包相关的关键字和其对应的意义如表 2-4 所示。

表 2-4　包的关键字

关键字	含义	备　注
import	引入包	使用其他包中的类是需要使用其关键字 用法：import＋包名
package	定义包	定义类所在的包 用法：package＋包名

第四是关于数据类型的关键字，什么是数据类型呢？数据类型是用来约束数据的解释。在编程语言中，常见的数据类型包括原始类型(如：整数、浮点数或字元)、多元组、记录单元、代数数据类型、抽象数据类型、参考类型、类以及函式类型。数据类型描述了数值的表示法、解释和结构，并以算法操作，或是物件在内存中的储存区，或者其他储存装置。罗列一些专业定义，简而言之就是数据的身份表示，如果你想对两个自然数进行计算，那么就应该使用 int 对数据进行定义，而后将需要计算的数值赋值给该变量。定义变量的基本操作为：关键字＋变量名。具体与数据类型相关的关键字和其对应的意义如表 2-5 所示。

表 2-5　数据类型的关键字

关键字	含义	备　注
byte	字节型	8bit
char	字符型	16bit
boolean	布尔型	即判断条件的真或假
short	短整型	16bit

续表

关键字	含义	备　注
int	整型	32bit
float	浮点型	32bit
long	长整型	64bit
double	双精度	64bit
void	无返回	public void A(){}其他需要返回的经常与 return 连用
null	空值	常常在判断对象或变量是否为空时使用
true	真	属于 boolean 类型的值
false	假	属于 boolean 类型的值

第五是与条件循环相关的关键字。条件循环几乎是每一门编程语言中均必不可少的功能，使用条件循环后，我们的程序拥有了初步判断和流程化的能力，能够为我们分析多种情况下的操作，通过设置的条件进行判断，程序可以实现选择性跳转的功能。具体与条件循环相关的关键字和其对应的意义如表 2-6 所示。

表 2-6　条件循环

关键字	含义	备　注
if	如果	用法：if(条件){执行语句}
else	否则，或者	else 是 if 条件为假时，为程序作下一步处理的关键字 用法：else{执行语句}
while	当条件为真时	当 while 中的条件为真时，程序会循环执行 while 体中的语句 用法：while(条件){执行语句}
for	当条件为真时	当 for 中的条件为真时，程序会循环执行 for 体中的语句。与 while 不同的是 for 可以自行为变量或条件进行计算操作，相比 while 在某些时候更为方便
switch	开关	switch 关键字就像是一个多 if 语句，你可以在 switch 体中设置多个条件满足某个条件，就会跳转至某个条件中的语句继续执行。 用法：switch(变量){ case 条件一： 　　执行语句 　　break; case …： 　　执行语句 　　break; }
case	返回开关里的结果	
default	默认	
do	运行	长与 while 连用
break	跳出循环	
continue	继续	中断本次循环，并开始下一次
return	返回	return 一个返回值类型
instanceof	是否相等	用于测试它左边的对象是否是它右边的类的实例，返回 boolean 类型的数据

最后是关于错误处理的关键字。在编写程序时，用户可能会因为粗心大意而写错了代码，这时候，编译器能够为我们发现语法错误，并及时纠正。但在程序运行时，由于用户或者其他因素的影响也可能会造成程序出错，这时候 Java 语言也提供了一个错误处理机制，通过这个错误处理，来保证遇到错误时，程序也能有相关的应对措施。具体与错误处理相关的关键字和其对应的意义如表 2-7 所示。

表 2-7 错误处理

关键字	意思	备注，常用
catch	处理异常	(1) try＋catch 程序的流程是：运行到 try 块中，如果有异常抛出，则转到 catch 块去处理。然后执行 catch 块后面的语句。 (2) try＋catch＋finally 程序的流程是：运行到 try 块中，如果有异常抛出，则转到 catch 块，catch 块执行完毕后，执行 finally 块的代码，再执行 finally 块后面的代码。如果没有异常抛出，执行完 try 块，也要去执行 finally 块的代码。然后执行 finally 块后面的语句。 (3) try＋finally 程序的流程是：运行到 try 块中，如果有异常抛出的话，程序转向执行 finally 块的代码。那么 finally 块后面的代码还会被执行吗？不会！因为你没有处理异常，所以遇到异常后，执行完 finally 后，方法就已抛出异常的方式退出了
try	捕获异常	同上
finally	有没有异常都执行	同上
throw	抛出一个异常对象	一些可以导致程序出问题的因素，例如书写错误，逻辑错误或者是 api 的应用错误等等。为了防止程序的崩溃就要预先检测这些因素，所以 java 使用了异常这个机制。 在 java 中异常是靠“抛出”也就是英语的“throw”来使用的，意思是如果发现异常的时候就把错误信息“抛出”
throws	声明一个异常可能被抛出	把异常交给其上级管理，自己不进行异常处理

2. 保留字

在 Java 中，const 和 goto 就是两个保留字——它们在 Java 中目前没有被使用，因此不具有意义，但是不能够被用作标识符。通过“保留”这个术语，它们可以在 Java 的未来版本中补充，而不需要“破坏”旧的 Java 源代码。不像预定义函数、方法和子程序，保留字不能被程序员定义，而前面那些的名称通常被归类于标识符，而不是保留字。

在日常开发的过程中，goto 的出场率其实极低，而且一直被人们所诟病，但由于多方面的原因依旧留下了这一保留字，因此，在此我们并不对 goto 这一保留字作详细的阐述，有心的读者可以通过自行查找相关资源进行学习。

学习过 C 或 C++ 的同学可能对 const 这一词较为熟悉，它被用于定义常量。但在 Java 语言中，我们并不使用保留字 const 对常量进行定义，而是使用 final 关键字对其进

行处理。而且在Java中类和方法被声明为final时不能被继承。const只作为一个保留字在Java语言中存在,所以我们了解即可。

关于保留字,其实在Java语言中使用的次数和频率极低,同学们只需要对其进行了解,知晓这一概念就可以了。

2.4 数据类型与变量

在上节内容中,我们了解了有关数据类型的关键字,对它们的使用也有了基本的掌握,在本节中,我们会深入地讲解关于数据类型和变量的知识点。Java基本类型共有八种,基本类型可以分为几类,分别为整型(int、short、long、byte)、浮点型(float、double)、字符型(char)、布尔型(boolean)。

2.4.1 整型

在我们日常学习中,总是会遇到需要进行加减乘除运算的时候,我们书写自然数、小数或者无理数时,只需使用我们人类自己能够理解的内容即可。但Java无法像人类一样极具智慧,并自行分辨出我们所输入的数字大小和类型,所以我们需要通过使用整型或其他不同类型的关键字来声明变量,使我们输入的数值能够被Java语言所理解,被计算机所理解。而在Java语言中有4种整型用于表示整数,正负均可。关于Java语言中的整型,具体的类型和其取值范围如表2-8所示。

表2-8 Java整型

类型	存储需要	取值范围	默认值
int	4字节	−2147483648～2147483647	0
short	2字节	−32768～32717	0
long	8字节	−2^63～2^63−1	0L
byte	1字节	−128～127	0

Java决定了每种基本类型的大小。这些大小并不随着机器结构的变化而变化。这种大小的不可更改正是Java程序具有很强移植能力的原因之一。

这四种整型也有以下的区别。

(1) byte类型用在大型数组中节约空间,主要代替整数,因为byte变量占用的空间只有int类型的四分之一。

例如:

```
byte a= 10
```

(2) Short数据类型也可以像byte那样节省空间。一个short变量是int型变量所占空间的二分之一。

例如:

```
short a= 100
```

(3) 一般地整型变量默认为 int 类型。

例如：

```
int a= 123
```

(4) long 数据类型在初始化时需在数字末尾加上“L”。“L”理论上不分大小写，但是若写成“l”容易与数字“1”混淆，不容易分辨。所以最好使用大写。

例如：

```
long a= 100000L
```

可以看到 byte 和 short 的取值范围比较小，而 long 的取值范围太大，占用的空间多，基本上 int 可以满足我们日常的计算了，而且 int 也是使用最多的整型类型了。在通常情况下，如果 Java 中出现了一个整数数字如 35，那么这个数字就是 int 型的(因为整数默认类型是 int)，如果我们希望它是 byte 型的，可以在数据后加上大写的 B：35B，表示它是 byte 型的，同样地，35S 表示 short 型，35L 表示 long 型。

长整型数值有一个后缀 L(如 1000000000L)。十六进制数值有一个前缀 0x(如 0xCAFE)。八进制有一个前缀 0，例如，011 对应八进制中的 9。很显然，八进制表示法比较容易混淆，所以建议最好不要使用八进制常数。

从 Java 7 开始，加上前缀 0b 就可以写二进制数。例如 0b1111 就是 15。另外，同样是从 Java 7 开始，还可以为数字字面量加下画线，如 1_000_000，不过这些下画线也只是为了让程序变得更易读懂，java 编译器会去除这些下画线。

2.4.2 浮点型

在上节开头，我们提到了在日常学习中，我们需要使用整数、小数等不同形式的数字来进行计算，在 Java 语言中，也提供了专门的用于对小数进行存储的类型。合理使用这些类型的关键字，可以使用 Java 语言的计算事半功倍。在 Java 语言中，浮点型用于存储带有小数的数据。关于 Java 语言中的整型，具体的类型和其取值范围如表 2-9 所示。

表 2-9　Java 浮点型

类型	存储需要	取值范围	默认值
float	4 字节	－3.40E＋38～＋3.40E＋38(有效位数为 6～7 位)	0.0f
double	8 字节	－1.79E＋308～＋1.79E＋308(有效位数为 15 位)	0.0d

注意：

(1) float 类型的数值后面必须加上 F 或者 f(例：12.3f)，否则该数值将被默认为 double 类型的数据。所以数值 12.3 等同于 12.3d。

(2) double 类型的数值精度是 float 类型的两倍，故有人称 double 类型为“双精度”、float 类型为“单精度”。double 的精度能满足的精度需求更广泛，所以不难想象在实际开发过程中 double 类型的使用频率要远高于 float 类型。

(3) 所有的浮点数计算都遵循 IEEE 754 规范。float、double 两种类型的最小值与

Float. MIN_VALUE、Double. MIN_VALUE 的值并不相同，实际上 Float. MIN_VALUE 和 Double. MIN_VALUE 分别指的是 float 和 double 类型所能表示的最小正数。也就是说存在这样一种情况，0 到 ± Float. MIN_VALUE 之间的值，float 类型无法表示；0 到±Double. MIN_VALUE 之间的值，double 类型无法表示。这并没有什么好奇怪的，因为这些范围内的数值超出了它们的精度范围。

2.4.3 char 型

在 Java 语言中，我们有时会需要存储一句话，例如在第 1 章中图书管理系统的欢迎语：欢迎来到图书管理系统。这样一句话在 Java 语言中就是一个字符串，我们可以使用 String 类型的变量对它进行存储。但这个字符串中的“欢”字却有两个意义，第一它是个字符串，第二它也是一个字符，字符在计算机内存中的存储是以编码的形式来完成的，所以，在编程中若想要对某个字符的编码进行获取和判断，我们就不能使用 String 类型的变量，而是应该使用 char 类型的变量，我们可以对这个 char 类型的变量进行数值对比和计算操作，这样在某些特定的情景下，既能够提高程序的运行速度，也能够快速地完成开发任务，降低开发难度。说了那么多 char 类型的作用，我们来介绍一下 char 类型。char 在 Java 中是 16 位的，因为 Java 使用的是 Unicode。同时 8 位的 ASCII 码包含在 Unicode 中，编码为 0～127。

char 可以使用以下的方式进行初始化。

(1) char c='c'; //'c'与"c"不同，'c'对应的是编码值为 99 的字符常量，而"c"是包含一个字符 c 的字符串。

(2) char c='汉'; //单引号中可以是单个汉字。

(3) char c=111; //整数。0～65535。十进制、八进制、十六进制均可。输出字符编码表中对应的字符。

(4) char c='\u0000'; //转义字符+数字。

除了正常的字符外，还有如下一些用于表示特殊字符的转义序列。具体的转义字符和其名称如表 2-10 所示。

表 2-10　特殊字符转义序列

转义序列	名　称	Unicode 值
\b	退格	\u0008
\t	制表	\u0009
\n	换行	\u000a
\r	回车	\u000d
\”	双引号	\u0022
\’	单引号	\u0027
\\	反斜杠	\u005c

2.4.4 布尔型

我们在使用 if 或 while 循环时，都需要为其设置条件，条件的真假影响了 if 和 while

的下一步操作。而条件的真假在 Java 中也有专门的布尔(boolean)类型来存储。布尔类型占 1 个字节,用于判断真或假(仅有两个值,即 true、false),默认值 false。布尔值与整型值之间不能进行转换。

我们在编程时,也有多种使用的形式,具体的使用形式如下:

(1) 直接赋值 boolean b1=false;

(2) 由条件表达式赋值 boolean b2=3>4;

(3) 由另一个 boolean 变量赋值 boolean b3=b1。

在编程中,你可能会发现 boolean 和 Boolean 两个关键字都可以对变量进行布尔类型的赋值。有什么区别吗? 其实深究其原理,Boolean 是 boolean 类型的封装类,它可以通过 new 的形式来生成一个 Boolean 类型的对象,并含有一些基本或实用的方法。有兴趣的话,你也可以自己探究一下。

2.4.5 变量

在本节的学习中,我们将具体阐述变量的使用规则和使用时的注意事项。在 Java 语言中,我们若想要使用某个类型的数据或者想要存储某个类型的数据,都需要将该数据类型定义为一个变量,通过对变量进行赋值操作,才能够对这个数值进行相应的操作和运算。变量就相当于货币,你必须有这个货币才能在相应的地方进行消费和功能使用。当然,必须要是类型相同的货币和消费场所才能进行这样的操作,否则,你可能会遇到很多"麻烦(类型错误)"。现在我们就来看看如何正确地使用变量。

变量定义的基本格式为:type+identifier=value; 或直接";"结尾。

若变量是全局变量,则格式应为:public+type+identifier=value; 或直接";"结尾。若变量是局部变量,则按照基本格式书写即可。

例如:

```
String text= "this is type of String"
```

在我们定义变量时,应该注意以下几点。

(1) 在 Java 程序设计中,每个声明的变量都必须有一个数据类型。声明一个变量时,应该先声明变量的类型,随后再声明变量的名字。

注意:不能使用未初始化的变量,否则系统将报错。

(2) 变量初始化只需要将变量名和相应取值或表达式分别放在等号(=)的左右。

(3) 可以将变量声明和变量初始化在一行中书写。

2.4.6 常量

常量是一种特殊的变量,它一旦被赋值后就不能更改。常量的这一特性可以提高代码的可维护性。例如,在项目开发时,我们需要指定用户的性别,此时可以定义一个常量 age,赋值为"20",在需要指定用户年龄的时候直接调用此常量,避免了由于用户的不规范赋值导致程序出错的情况。

常量的声明方式也与变量相似，只是在数据类型前多了一个修饰符 final。一般常量名为全大写。在 Java 中，在变量声明中加入 final 关键字代表常量，加入 static 关键字代表类变量。一般情况下，我们把 static 与 final 结合起来声明一个常量。

在常量定义时也有一些基本的注意事项。

(1) Java 常量定义的时候，就需要对常量进行初始化。也就是说，必须要在常量声明时对其进行初始化。与局部变量或者成员变量不同。当对常量进行初始化之后，在应用程序中就无法再次对这个常量进行赋值。如果强行赋值的话，数据库会跳出错误信息，并拒绝接受这一个新的值。

(2) final 关键字使用的范围。这个 final 关键字不仅可以用来修饰基本数据类型的常量，还可以用来修饰对象的引用或者方法。如数组就是一个对象引用。为此，可以使用 final 关键字来定义一个常量的数组。这就是 Java 语言中一个很大的特色。一旦一个数组对象被 final 关键字设置为常量数组之后，它只能够恒定地指向一个数组对象，无法将其改变，指向另外一个对象，也无法更改数组(有序数组的插入方法可使用的二分查找算法)中的值。

(3) 需要注意常量的命名规则。不同的语言，在定义变量或者常量的时候，都有自己的一套编码规则。这主要是为了提高代码的共享程度与提高代码的易读性。在 Java 常量定义的时候，也有自己的一套规则。如在给常量命名的时候，一般都用大写字符。在 Java 语言中，大小写字符是敏感的。之所以采用大写字符，主要是跟变量进行区分。虽然说给常量命名时采用小写字符，也不会有语法上的错误。但是，为了在编写代码时能够一目了然地判断变量与常量，最好还是能够将常量设置为大写字符。另外，在常量中，往往通过下画线来分隔不同的字符。而不像对象名或者类名那样，通过首字符大写的方式来进行分隔。这些规则虽然不是强制性的规则，但是为了提高代码友好性，方便开发团队中的其他成员阅读，这些规则还是需要遵守的。没有规矩，不成方圆。

总之，Java 开发人员需要注意，被定义为 final 的常量需要采用大写字母命名，并且中间最好使用下画线作为分隔符来连接多个单词。定义 final 的数据不论是常量、对象引用还是数组，在主函数中都不可以改变。否则，会被编辑器拒绝并提示错误信息。

2.5 运算符、表达式与控制语句

在本节的学习中，我们会详细地了解运算符、表达式和控制语句，使大家对这几个概念有完备的了解并能熟练使用。在前边几节讲解关键字和变量时，我们就听到过对变量进行计算和使用控制语句。大家可能对这两个知识点都有所了解，但不一定能够正确使用，也不一定能够在合适的时机使用它们，通过学习本节内容，你将会对其更为了解和熟悉。

2.5.1 运算符

在前几节的内容中，我们就提及过使用 Java 语言进行数值计算或者计算机底层的位运算。我们在现实生活中进行计算的基础是，已经理解每个计算符号所代表的意义，同样的 Java 语言也需要理解这些运算符才能为我们提供相对应的计算。Java 的设计者已经

为我们完成了这些运算符的底层实现,我们只需要使用与实际计算符号相同的符号就可以在 Java 语言中进行数值计算的需求。除了个别几个在现实生活计算中不涉及的符号,其他的运算符都和数学中的运算符保持一致。具体的 Java 算术运算符的描述和定义如表 2-11 所示。

表 2-11　Java 算术运算符

操　作　符	描　　述
+	加法—相加运算符两侧的值
−	减法—左操作数减去右操作数
*	乘法—相乘操作符两侧的值
/	除法—左操作数除以右操作数
%	取模—左操作数除以右操作数的余数
++	自增:操作数的值增加 1
−−	自减:操作数的值减少 1

这里需要注意的是:一般自增和自减操作是在循环中使用,我们将在接下来的内容中介绍关于循环的使用。

2.5.2　关系运算符

前边的章节内容中,我们有提到过关于 if 语句的使用,if 语句是通过判断某个设置的条件来决定应该执行哪一步的操作。而我们设置的这些条件则需要使用到一些关系运算符来对该条件是否满足进行判断。正是有了这一系列的关系运算符,我们的 Java 语言条件判断才得以完善,才能够使我们的编程变得更简洁。具体的 Java 关系运算符的描述如表 2-12 所示。

表 2-12　Java 关系运算符

运　算　符	描　　述
==	检查如果两个操作数的值是否相等,如果相等则条件为真
!=	检查如果两个操作数的值是否相等,如果值不相等则条件为真
>	检查左操作数的值是否大于右操作数的值,如果是,那么条件为真
<	检查左操作数的值是否小于右操作数的值,如果是,那么条件为真
>=	检查左操作数的值是否大于或等于右操作数的值,如果是,那么条件为真
<=	检查左操作数的值是否小于或等于右操作数的值,如果是,那么条件为真

注意:在“==”的时候,若比较的两个变量是 String 类型的值,一定要使用 equals() 这个函数进行判等,否则,“==”比较的只是两个变量的地址,而不是其内容。在比较例如整型、浮点型的数值时,则可以使用“==”进行判等操作。

2.5.3　逻辑运算符

在上一节的内容中,我们了解了如何使用条件运算符在 Java 中处理条件的判断问题。在实际的编程中,很多时候判断条件并不唯一。例如,我想吃的食物是一碗小面而且

价格不能超过9元，那么把这个条件告诉电脑，用一个条件是没有办法实现的。因为食物是否是小面是一个条件，价格是否低于9元又是另一个条件。所以当我们实现一个多条件的判断语句时，如果能使用连接词“并且”和“或者”，就能够帮我们解决这个问题了。幸运的是，Java语言中，提供了这一系列的关键词，帮助我们在遭遇到多个条件时，能够简化为一个判断语句为我们处理，这样在代码和逻辑上都会变得简洁。具体的Java逻辑运算符的描述如表2-13所示。

表2-13 Java逻辑运算符

运 算 符	描 述	例 子
&&	称为逻辑与运算符。当且仅当两个操作数都为真，条件才为真	A为假，B为真，(A&&B)为假
‖	称为逻辑或操作符。如果任何两个操作数任何一个为真，条件为真	A为假，B为真，(A‖B)为真
!	称为逻辑非运算符。用来反转操作数的逻辑状态。如果条件为true，则逻辑非运算符将得到false	A为假，B为真，!(A&&B)为真

注意：使用“&&”时，一假则假——一个条件为假，整个条件判断均为假，且当第一个条件判断为假时，则不会在继续判断下一个条件的真假；使用“‖”时，一真则真——一个条件为真，则整个条件判断均为真，且当第一个条件判断为真时，则不会再继续判断下一个条件的真假。

2.5.4 赋值运算符

早在2.4.5节我们就介绍了变量的概念，在定义变量和赋值的操作中，我们使用了“=”来为变量进行赋值操作。其实，在Java中还有很多种为变量进行计算的赋值运算符，通过这些运算符，我们能够简化书写程序的代码，得到更高的可读性。具体的Java赋值运算符如表2-14所示。

表2-14 Java赋值运算符

运 算 符	描 述	例 子
=	将右操作数的值赋给左侧操作数	C=A+B将把A+B得到的值赋给C
+=	把左操作数和右操作数相加赋值给左操作数	C+=A等价于C=C+A
-=	把左操作数和右操作数相减赋值给左操作数	C-=A等价于C=C - A
*=	把左操作数和右操作数相乘赋值给左操作数	C*=A等价于C=C*A
/=	把左操作数和右操作数相除赋值给左操作数	C/=A等价于C=C/A
(%)=	把左操作数和右操作数取模后赋值给左操作数	C%=A等价于C=C%A
<<=	左移位赋值运算符	C<<=2等价于C=C<<2
>>=	右移位赋值运算符	C>>=2等价于C=C>>2
&=	按位与赋值运算符	C&=2等价于C=C&2
^=	按位异或赋值操作符	C^=2等价于C=C^2
|=	按位或赋值操作符	C|=2等价于C=C|2

2.5.5 运算符优先级

我们在学习数学的基础计算时，了解了如何对数值进行加减乘除的运算，而且，当我们使用不同的运算时，我们需要考虑到乘法和除法先进行，加法减法靠后计算。但当我们使用了不同等级的括号后，情况又会发生改变。这就是数学中的优先级。当然，在 Java 语言中，程序也遵循相同的规则。在一个表达式中可能包含多个有不同运算符连接起来的、具有不同数据类型的数据对象；由于表达式有多种运算，不同的运算顺序可能得出不同结果，甚至出现错误运算错误，因为当表达式中含多种运算时，必须按一定顺序进行结合，才能保证运算的合理性和结果的正确性、唯一性。为了保证操作和计算结果正确，我们来学习 Java 运算符优先级规则。具体的规则和顺序如表 2-15 所示。

表 2-15 Java 运算符优先级

<table>
<tr><th>优先级</th><th>操作符</th><th>关联性</th></tr>
<tr><td>1</td><td>()[].(点操作符)</td><td>从左到右</td></tr>
<tr><td>2</td><td>++－!～</td><td>从右到左</td></tr>
<tr><td>3</td><td>*/%</td><td>从左到右</td></tr>
<tr><td>4</td><td>+－</td><td>从左到右</td></tr>
<tr><td>5</td><td><< >> >>></td><td>从左到右</td></tr>
<tr><td>6</td><td>> >= < <= instanceof</td><td>从左到右</td></tr>
<tr><td>7</td><td>== !=</td><td>从左到右</td></tr>
<tr><td>8</td><td>&</td><td>从左到右</td></tr>
<tr><td>9</td><td>^</td><td>从左到右</td></tr>
<tr><td>10</td><td>|</td><td>从左到右</td></tr>
<tr><td>11</td><td>&&</td><td>从左到右</td></tr>
<tr><td>12</td><td>‖</td><td>从左到右</td></tr>
<tr><td>13</td><td>?:</td><td>从右到左</td></tr>
<tr><td>14</td><td>= += -= *= /=%=
>>= <<=&= ^= |=</td><td>从右到左</td></tr>
</table>

注意：

(1) 该表中优先级按照从高到低的顺序书写，也就是优先级为 1 的优先级最高，优先级为 14 的优先级最低。

(2) 结合性是指运算符结合的顺序，通常都是从左到右。从右向左的运算符最典型的就是负号，例如 3+－4，则意义为 3 加－4，符号首先和运算符右侧的内容结合。

(3) instanceof 作用是判断对象是否为某个类或接口类型，后续有详细介绍。

(4) 注意区分正负号和加减号，以及按位与和逻辑与的区别。

2.5.6 控制语句

在本章的前几节中，均提及过关于控制语句的知识点，想必同学们或多或少都对它们有了一个最基础的印象，或者说是自己的理解。那么，在本节的学习中，我们将重点和系统地对这个知识点进行学习和练习。学习任何一种语言都需要学习它的语法和词汇，这

样我们才能说出规范的语句。学习语言的过程就像是建造一栋大楼,词汇是砖块,语法是框架(即语言的框架),而控制语句就像是 Java 语言中的语法。Java 中有顺序控制、条件控制、循环控制。顺序控制就是从头到尾依次执行每条语句操作,不能进行判断和选择。顺序结构适应人类的思维模式。但为了应对更多的场景,就出现了条件控制和循环控制。

1. 条件控制

条件控制语句主要包含两个关键字 if 和 switch,其中包含多种分支结构。

1)单分支结构

单分支结构的特点是只会对“是”或“否”两种情形中的其中一种做出判断和操作。例如,询问你肚子饿吗?这时,我们就可以使用单分支结构来对这个条件进行处理。在单分支结构中设置的条件则需要我们自己根据实际情况进行编写。至于在执行单分支结构后需要执行的语句,我们则按照自己的需要来编写。单分支结构一般可以用在某些函数开头,判断某个操作或变量是否符合运行后续操作的条件,若不符合则直接返回失败,防止程序运行异常。其他的情况,在实际的编程中,读者可以适当适时使用。单分支结构的书写形式如下:

```
if (表达式) {
      方法体
}
```

表达式的结果是一个布尔值,如果是 true,直接进入 if 的方法体中,如果结果为 false,则跳过 if 的方法体,继续执行。

示例代码

```
public class Test {
    public static void main(String[] args) {
        int num = 1;
         //当 num 等于 1 时,执行 if 中的打印语句
        if (num == 1) {
            System.out.println("这个数字是 1");
        }
    }
}
```

2)双分支结构

双分支结构的特点是“非黑即白”,不是对的就一定是错的。例如,询问你是男生还是女生?那么答案就只有两种,这时,我们就可以使用双分支结构来对这个条件进行处理。至于,在执行双分支结构后需要执行的语句,我们则按照自己的需要来编写。双分支结构的书写形式如下:

```
if(表达式){
      方法体
}else{
```

```
        方法体
    }
```

表达式的结果是一个布尔值，如果是 true，直接进入 if 的方法体中，如果结果为 false，则直接执行 else 中的方法体，然后跳出 else。

示例代码

```
public class Test {
    public Test() {
        int num = 1;
        if (num == 1) {
            System.out.println("这个数是 1");
        }else{
            System.out.println("这个数不是 1");
        }
    }
}
```

3）多分支结构

多分支结构的特点是支持多种情况。从本质上来说，多分支结构就是将 if 与 else 嵌套使用。使用多分支结构的好处就是在一个代码块中可以判断多种情形。请谨慎使用，过多地使用多条件判断也会在一定情况下影响到整个程序的运行速度。多分支结构的书写形式如下：

```
if(表达式 1) {方法体 1}
else if(表达式 2) {方法体 2}
else if(表达式 3) {方法体 3}
……
else if(表达式 n) {方法体 n}
else {方法体 n+1}
```

从上到下不断判断表达式，若符合表达式 n 为 true 则执行方法体 n 后跳出，反之则继续判断表达式 n＋1，如若表达式都为 false，则执行 else 中的方法体后跳出。

4）switch 语句

switch 关键字其实就是一个升级版的多分支结构。通过对一个输入的值获取，即可在其内部设置判断条件，通过判断条件的真假来决定执行的语句。但这个判断条件必须将其结果直接给出。同时，你也可以设置默认情况，当所有条件都不满足时，默认执行的操作。switch 就像是一个开关一样，当满足条件时，Java 语句就把开关的档位调至该条件所在的 case 处，完成此处实现的功能。switch 语句结构的书写形式如下：

```
switch(获取值){
    case 1:
        执行内容 1;
        break;
    case 2:
```

```
        执行内容 2;
        break;
    case n:
        执行内容 n;
        break;
    default:
        执行默认内容;
        break;
}
```

注意：switch 语句将从与 choice 匹配的 case 处开始执行，遇到 break 后跳出。如果没有匹配的 case，则执行 default 后的代码。

示例代码

```
public class Test {
    public static void main(String[] args) {
        int num = 1;
        switch(num) {
        case 1:
            System.out.println("数字为 1");
            break;
        case 2:
            System.out.println("数字为 2");
            break;
        default:
            System.out.println("既不是 1 也不是 2");
            break;
        }
    }
}
```

通过以上示例代码我们可以得知，当 num = 1 时，switch 会执行 case 1 后的打印语句。

2. 循环结构

循环语句在程序设计中用来描述有规则重复的流程。在实际的程序中，存在很多需要重复执行的流程，为了简化这些重复的执行流程，在程序设计语言中新增了该类语句。

在学习循环语句时，最重要的就是发现流程的规律，然后再用程序设计语言将该规律描述出来，从而实现程序要求的流程。

循环语句是流程控制中最复杂，也是最有用、最难掌握的语句，在最初接触时，首先要熟悉基本的语法，然后需要能够快速观察出流程的规律，这种观察能力需要依靠大量的阅读和编写程序来进行培养，这就是基本的逻辑思维，然后将该规律描述出来即可。所以在学习循环语句时，学习语法只是基本的内容，更重要的是培养自己观察规律的能力，这个才是学习循环语句时的真正难点，也是重点。

循环结构主要有 3 个关键字：for、while、do-while。下面将为大家依次介绍。

1）for 循环

在前边的内容中，我们也提及过循环这一个概念。通过使用循环我们能够将数组或者其他形式的数据组按顺序或按规则将其取出。那么，我们就来了解一下 for 循环的基本概念。它的一般形式为：for(<初始化>；<条件表达式>；<增量>) 语句；初始化总是一个赋值语句，它用来给循环控制变量赋初值；条件表达式是一个关系表达式，它决定什么时候退出循环；增量定义循环控制变量每循环一次后按什么方式变化。这三个部分之间用"；"分开。例如：for(i=1；i<=10；i++) 语句；上例中先给"i"赋初值 1，判断"i"是否小于等于 10，若是则执行语句，之后 i 的值增加 1。再重新判断，直到条件为假，即 i>10 时，结束循环。for 循环的书写形式如下：

```
for(1部分;2部分;3部分)
{
    循环体
}
```

1 部分用于初始化计数器，2 部分是每一次执行新一轮循环体前必须通过的检测条件，3 部分的作用是更新计数器。

示例代码

```
public class Test {
    public static void main(String[] args) {
        for(int i=0; i<100; i++) {
            System.out.println(i);
        }
    }
}
```

通过这段示例代码，我们可以了解 for 循环的基本使用，程序执行结果则是打印 0～99 这 100 个数字。

2）while 循环

while 循环是 Java 语言中，最为基础循环语句之一。我们可以通过使用 while 循环帮我们判断或者处理一系列相同属性或条件的事务和代码。其实，根据 while 的中文意义理解，我们可以更加形象和简单地理解循环。while 关键字的中文意思是"当……的时候"，也就是当条件成立时循环执行对应的代码。while 语句是循环语句中基本的结构，语法格式比较简单。while 循环的书写形式如下：

```
while(循环条件)
{
    循环体
}
```

在每次执行循环体之前判断是否符合循环条件。若符合则执行循环体，再次判断循环条件；若不符合循环体，则直接跳过 while 循环。注意：需要在循环体中改变循环条件中涉及的变量，否则将陷入死循环。

示例代码

```
public class Test {
    public static void main(String[] args) {
        int num = 0;
        while(num < 100) {
            System.out.println(num);
            num ++;
        }
    }
}
```

以上示例代码，是使用 while 循环来打印 0～99 这 100 个数字，与 for 循环小节示例代码实现的功能一致，只是使用的关键字和实现的方法不同。

3）do-while 循环

do-while 语句由关键字 do 和 while 组成，是循环语句中最典型的“先循环再判断”的流程控制结构，这个和其他 2 个循环语句都不相同。在 do-while 语句中，循环体部分是重复执行的代码部分，循环条件指循环成立的条件，要求循环条件是 boolean 类型，值为 true 时循环执行，否则循环结束，最后整个语句以分号结束。当执行到 do-while 语句时，首先执行循环体，然后再判断循环条件，如果循环条件不成立，则循环结束，如果循环条件成立，则继续执行循环体，循环体执行完成以后再判断循环条件，依次类推。do-while 循环的书写形式如下：

```
do{
    循环体
}while(循环条件);
```

do-while 循环与 while 循环有一点不同，do-while 循环在第一次执行循环体之前不会判断循环条件。

示例代码

```
public class Test {
    public static void main(String[] args) {
        int num = 0;
        do {
            System.out.println(num);
            num ++;
        }while(num < -1);
    }
}
```

通过以上示例代码我们可以了解到，do-while 循环不同于 while 循环之处在于，无论是否满足条件，在运行程序时，do-while 循环均会先执行一次循环中的内容，再进行判断，而 while 循环则是相反的。

2.6　数　　组

在本节中我们将对数组进行学习。可能大家对这个概念还是很陌生的。数组顾名思义就是与数字有关的小组。什么是小组呢？例如，现在班级中有五个同学，这五个同学都是男生，那么我们可以将他们排好队，安排在一个小组中并为他们从 0～4 进行编号，那么这五个同学就可以看作是在一个数组中。当你叫 3 号同学时，第四个男生就会有所反应。这就是数组。下面我们看看具体的关于数组的描述。

数组是一种数据结构，用来存放同一类型元素的集合。数组可以直接通过整型下标快速地访问和设置数组中的每一个值。在有许多数据的时候显得尤其的便捷。数组是一个简单的复合数据类型，它是一系列有序数据的集合，它当中的每一个数据都具有相同的数据类型，我们通过数组名加上一个不会越界下标值来唯一确定数组中的元素。

1. 数组的声明

我们若想要使用数组，则需要在相应的位置进行声明，告知编译器和 Java 语言你将要使用的数组数据类型是什么？数组名称是什么？这样当你想将数组中的数据取出时，Java 语言才能够判断应该对哪个数组进行操作。当然，数组的声明有两种方式。第一种声明方式是 C/C++风格的，这是为了让 C/C++的程序员快速地适应 Java 语言。两种方式都有同样的效果，具体使用哪种就由你决定。数组声明的书写形式如下：

```
dataType [] arrayName;
dataType arrayName [];
```

2. 数组的初始化

对数组进行了声明，则必定有对数组进行初始化的相关操作。声明是告知 Java 语言有某个数组对象的存在，初始化则是操作 Java 语言对数组的内容和大小进行确定。具体的数组初始化书写形式如下：

```
arrayName = new dataType[size];
```

Java 数组的初始化需要用到 new 操作符。当然，如果你想要再简单点，那么还可以将数组声明与数组初始化放在一起。

```
dataType [] arrayName = new dataType[size];
dataType [] arrayName = {value0,value1,value2, … ,valueN};
```

示例代码

```
public class Test {
    public static void main(String[] args) {
        int[] array = {1, 2, 3, 4};
         //输出所有数组元素
         for (int i = 0; i < array.length; i++) //数组的下标是从 0开始的
        {
            System.out.println(array[i] + " ");
```

```
        }
         //输出所有元素的总和
        double total = 0;
        for (int i = 0; i < array.length; i++)
        {
            total += array[i];
        }
        System.out.println("Total is " + total);
         //查找最大元素
        double max = array[0];
        for (int i = 1; i < array.length; i++)
        {
            if (array[i] > max) max = array[i];
        }
        System.out.println("Max is " + max);
    }
}
```

以上的示例代码展示了对数组进行的一些基本操作，例如对数组内容的遍历、数组元素的求和、查找最大元素。在数组中存储的元素并不都是按顺序摆放的，如果我们需要得到一个有序的数组，则需要我们自行使用各种不同的排序算法来对该数组进行排序操作，每种排序算法都有自己的优缺点，具体的算法例如堆排序、快速排序等，都是值得我们在课余时间了解和学习的，有兴趣的同学可以以此为契机，进一步了解关于数据结构的相关知识。

同样地，数组不仅仅可以用于存储多个整型或浮点型的数字，它也可以用于存储自定义的对象或者其他基本数据类型的变量。

3. 多维数组

上边我们介绍了数组的声明和初始化，但那仅仅只是一维数组的情况。而在 Java 语言中，面对诸多数学问题时，例如向量矩阵等，使用一维数组是远远不够的。所以，多维数组的出现，为计算多维向量和矩阵带来了便利。多维数组其实就是"数组的数组"。例如二维数组就是一个数组的所有元素都是一个一维数组。和一维数组是一样的，多维数组也需要在使用时进行声明和初始化。具体的多维数组初始化书写形式如下：

```
dataType arrayName = new datatype[Row][Column];
dataType arrayName = {{value1,value2,value3},{value1,value2,value3}};
```

dataType 可以是基本数据类型，也可以是复合数据类型。Row 是行数，Column 是列数。在这里复合数据类型指的是我们自己定义的类或是其他的非常规数据类型。

示例代码

```
public class Test {
    public static void main(String[] args) {
        int a[][] = {{1,2,3},{4,5,6},{7,8,9}}; //这是一个二行三列的二维数组
```

```
            //遍历数组元素
            for (int x = 0;x < 3;x ++)
            {
                for(int y = 0;y < 3;y ++)
                {
                    System.out.println(a[x][y]);
                }
            }
        }
}
```

以上的示例代码为我们演示了如何对二维数组进行遍历。

通过对数组的学习,我们了解到如何初步对多个相同数据类型的值进行存储。在本章学习中,我们应该着重理解数组的含义,并且思考其使用的场景。同时,应该注意数组的下标范围,第一个元素的下标为0,最后一个元素的下标为数组长度－1。谨记这点,你能在编程中减少出现数组越界等相关问题的可能性。

2.7　基本输入输出

在本章的学习中,我们一直在学习如何使用Java语言来对已经在代码中预留好的数值或者字符串进行处理。相当于对于计算机而言,我们所有的输入都必须在编译.java文件之前就完成。这样做无可厚非,毕竟计算机只不过是用来为我们提供计算和处理信息的工具而已。可是,如果我们需要不停地修改程序中的参数或者通过编译器来输入需要计算机进行处理的信息,这样对于程序员而言可能是一件容易的事,但如果你编写的程序要给一个非程序员使用,这将是个大麻烦。首先,并不是每个人的电脑中都会有编译器,也不是每个人都了解和学习过Java语言,这样做对于业余人士来说十分不公平。而且,效率也十分低下。Java语言的设计者早已为我们考虑过这一问题的解决方法——提供可以获取电脑键盘输出的接口。

Java开发人员通过调用这一系列的接口,即可让程序在运行时,自动获取用户的输入,而不需要每次都自行更改变量的值来重新编译计算得到答案。这也是我们在学习Java过程中,第一次与计算机进行交互。在Java语言中,所提供的接口和类来获取用户输入一共有三种形式。这几种形式的使用就像浴室里的花洒,如果我们不安装花洒,那么用户的输入就是一条水流,当我们安装了花洒,那么水流的样式和作用就会有所改变。花洒的样式不同,水流的样式也会跟着发生改变。对于用户输入也是同样的道理,使用不同的形式,就会有不同的效果。下面我们将介绍三种形式,这三种形式分别是:按照字节来进行读取、按照字符串来进行读取、按照输入行的内容进行读取。

1. 按照字节读取输入

按照字节读取输入是最为简单的一种读取用户输入的方式。我们通过使用Java中的System.in.read();语句来对用户键盘输入进行读取。按照字节读取输入时,我们每次只能够从用户输入中读取一个字符的字节数。无论你输入多少个字符,Java默认读取

第一个字符的字节，并且当用户输入完毕后，按下回车键即可完成读取。通过如下的代码，我们可以基本地使用代码来获取用户的输入。

示例代码

```
import java.io.IOException;

public class Test {
    public static void main(String[] args) throws IOException {
        System.out.print("请输入 :");
        char i = (char) System.in.read();
        System.out.println("你的输入是 :"+i);
    }
}
```

注意：在讲解关键字的章节中，我们就区分过关于字符和字符串的不同。我们再次回忆一下，字符使用的关键字是 char，定义时使用单引号，只能存储一个字符。字符串则是使用关键字 String，定义时使用双引号，可以存储一个或多个字符。它们最大的区别在于 char 是基础的数据类型，而 String 是一个类，具有对象的各项特征。

2. 按照字符串读取

按照字符串来进行读取，我们可以将其理解为按照字符读取的升级版。用户在控制台中输入一个字符串，Java 语言运行相关的代码来进行读取。按照字符串读取主要是用到了 BufferedReader 类和 InputStreamReader 类。通过创建 BufferedReader 这个对象，调用 readLine()这一方法，我们即可读取用户输入的字符串内容。和按字符输入一样，当用户按下 Enter 键时，代表结束输入。具体的实现代码如下所示。

示例代码

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Test {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String str = null;
        System.out.println("请输入 :");
        str = br.readLine();
        System.out.println("你输入的是 :"+str);
    }
}
```

3. 按照输入内容读取

在上文中，我们了解和学习了如何将用户输入的字符串进行读取。大家对此也一定印象深刻，因为它能够完整地读取用户的整行输入。这样，为我们的编程和读取带

来了极大的便利。但在实际的应用场景中，只拥有读取字符串的功能往往是不够的，当输入的字符串中或是在实际的应用场合中，涉及整数、浮点型数值的输入时，使用功能更为强大和全面的 Scanner 类对用户输入进行读取，可以省去许多由于字符串转换带来的麻烦，由 Java 语言为你处理转换问题，何乐不为呢？下面我们就介绍一下 Scanner 类的基本用法。

与 BufferedReader 类相同，首先创建一个 Scanner 类的对象，其次，我们使用不同的 next 系列函数对不同的输入数据类型进行读取，从而免去自行转换带来的不便。与以上两种获取输入的方式相同，在输入完成后，按下 Enter 键结束。具体的实现代码如下所示。

示例代码

```
import java.util.Scanner;

public class Test {
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        System.out.println("请输入你的年龄:");
        int age = sc.nextInt();
        System.out.println("请输入你的姓名:");
        String name = sc.nextLine();
        System.out.println("请输入你的工资:");
        float salary = sc.nextFloat();
        System.out.println("你的信息如下:");
        System.out.println("姓名:"+name+"\n"+"年龄:"+age+"\n"+"工资:"+ salary);
    }
}
```

通过示例代码，我们可以很清晰地看到 Scanner 类支持对于多数据类型输入的读取，我们在使用后，可以轻松将用户的输入转换为对应的数据类型，减少操作的步骤，提高代码效率。

2.8　一个单机版、控制台、只有一个类的图书管理系统 V2.0

2.8.1　运行效果图

通过学习本章的知识点，完成一个单机版、控制台、只有一个类的图书管理系统 V2.0 后的系统运行效果如图 2-3 所示。

2.8.2　类结构示意图

本章单机版、控制台的图书管理系统 V2.0 只有一个类，通过扩展不同的功能函数来实现图书管理系统的增、删、改、查功能。增、删、改、查功能分别对应的函数为 addBook()、deleteBook()、changeBook()、searchBook()。本章的图书管理系统类结构示

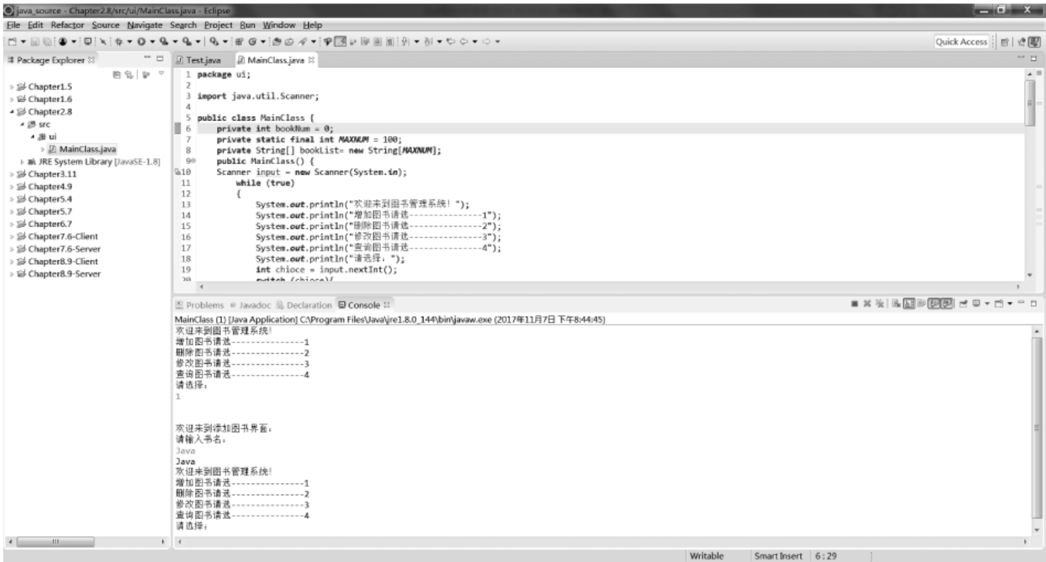


图 2-3　单机版、控制台、只有一个类的图书管理系统运行截图

意图如图 2-4 所示。



图 2-4　MainClass 主类结构图

2.8.3　代码实现

本章图书管理系统仅有一个类,代码如下:

Main.java

```java
package ui;

import java.util.Scanner;
```

```

public class MainClass {
        private int bookNum = 0;
        private static final int MAXNUM = 100;
        private String[] bookList= new String[MAXNUM];
        public MainClass() {
        Scanner input = new Scanner(System.in);
            while (true)
            {
                System.out.println("欢迎来到图书管理系统!");
                System.out.println("增加图书请选---------------1");
                System.out.println("删除图书请选---------------2");
                System.out.println("修改图书请选---------------3");
                System.out.println("查询图书请选---------------4");
                System.out.println("请选择:");
                int chioce = input.nextInt();
                switch (chioce){
                case 1:
                    addBook();
                    break;
                case 2:
                    deleteBook();
                    break;
                case 3:
                    changeBook();
                    break;
                case 4:
                    searchBook();
                    break;
                default:
                    break;
                }
            }
      }

      public void addBook()
      {
            Scanner input = new Scanner(System.in);
            System.out.println("\n\n欢迎来到添加图书界面:");
            System.out.println("请输入书名:");
            String bookname = input.nextLine();
            bookList[bookNum++] = bookname;
            System.out.println(bookList[0]);
      }
```

```

    public void deleteBook()
    {
        Scanner input = new Scanner(System.in);
        System.out.println("\n\n欢迎来到删除图书界面:");
        System.out.println("请输入需要删除的书名:");
        String bookname = input.nextLine();
        for(int i=0;i<bookNum;i++)
        {
            if(bookList[i].equals(bookname))
            {
                bookList[i]=bookList[i+1];
            }
        }
    }

    public void changeBook()
    {
         //具体代码由同学们自行编写
    }

    public void searchBook()
    {
         //具体代码由同学们自行编写
    }

    public static void main(String[] args) {
        new MainClass();
    }
}
```

第 3 章

面 向 对 象

现在，我们对 Java 的基础语法有了一定的了解，是时候向前迈进一步了，熟悉面向对象这一概念，对于理解 Java 编程大有裨益。在本章，我们也会详细讲解面向对象的相关概念。

3.1 本 章 任 务

理论任务：理解类、对象、继承、多态这四个基本概念，学会使用类，使用对象对数据进行处理。

实践任务：在第 2 章的基础之上，本章我们将定义一个 Book 类，将图书的信息保存到 Book 类的对象中，并使用 Book[]这一对象数组来对书籍进行增、删、改、查操作。从功能界面上来看，本章和上一章界面基本类似，但是完成了从 1 个类到 2 个类的过渡，初步建立了面向对象的概念。

3.2 面向对象基本概念

不知道读者是否还记得，在第 1 章的时候，我们举过一个“张三是个见人说人话见鬼说鬼话的人”的例子，这个例子是这样的：

用面向对象的思想就是：zhangsan. talk(person)或者是zhangsan. talk(ghost)。

用面向过程的思想就是：talk(zhangsan，person)或者是talk(zhangsan，ghost)。

其实，面向对象的思想就是从张三这个“实体”或者说“对象”出发，而面向过程则从说话这个动作出发。面向对象把解决的问题按照一定的规则划分为多个独立的对象，然后通过调用对象的方法来解决问题；面向过程就是分析解决问题所需要的步骤，然后用函数把这些步骤一一实现，使用的时候依次调用就可以了。

面向对象是一种符合人类思维习惯的编程思想，它所具有的特点主要可以概括为封装性、继承性和多态性。封装是面向对象的核心思想，将对象的属性和行为封装起来，不需要让外界知道具体实现细节，这就是封装思想。继承主要描述的是类与类之间的关系，通过继承，可以在无须重新编写原有类的情况下，对原有类的功能进行扩展。多态指的是在程序中允许出现重名现象，它指在一个类中定义的属性和方法被其他类继承后，可以具有不同的数据类型或表现出不同的行为。在后面的小节中笔者将详细叙述面向对象的这三个特点，这里只是写出了简单的概念。

面向对象程序设计(Object-Oriented Programming,OOP)是种具有对象概念的程序编程范型,同时也是一种程序开发的抽象方法。对象指的是类的实例,OOP将对象作为程序的基本单元,将程序和数据封装其中,以提高软件的重用性、灵活性和扩展性。对象里的程序可以访问、修改与对象相关联的数据;而对象外的程序则根据不同的访问权限有不同的访问方式。

Java是完全面向对象的,可以说Java编程实质就是构建类的过程。在前两章的学习中,我们能够看到即使是main()函数也只能放在类中运行,这也体现了Java是完全面向对象的编程语言。在后续的学习中,读者也会更加深刻地体会到Java编程的这一思想。接下来的小节将围绕着面向对象的几个特征来讲解Java这门编程语言。

3.3 类与对象

3.3.1 类

类(class)是具有相同特性和行为的对象的抽象,和int、char等数据类型类似,类也是一个数据类型,只不过它要复杂一些。类具有属性,它是对象的状态的抽象,用数据结构来描述类的属性。类具有操作,它是对象的行为的抽象,用操作名和实现该操作的方法来描述。对象的抽象是类,类的具象化就是对象。打个比方,有一个类是“书”,而有一本书是“Java程序设计教程”,“Java程序设计教程”就是“书”这个类的对象。由类构造对象的过程称为创建类的实例(instance)或者将类实例化。

由于Java是纯面向对象的语言,所以全部的Java代码都位于类的内部。在Java中,即使main()函数也是包含在类中的。

示例代码

```
public class Book {
    private String bookname;
    private String author;
    private int price;

    public Book(String bookname, String author, int bookPrice)
    {

        this.bookname = bookname;
        this.author = author;
        this.price = bookPrice;
    }
}
```

上述代码定义了一个Book类,这个类中有三个私有的成员变量(bookname、author、price),有一个带有三个参数的构造器(constructor)。通常情况下,我们将成员变量声明为私有的,具体原因将在3.4节中讲到。

3.3.2 对象

对象(object)是类的实例化。类相当于一个模板,对象就是按照这个模板生产出来的产品。

对象可以是人们要进行研究的任何事物,它不仅能表示具体的事物,还能表示抽象的规则、计划或事件。对象具有状态,一般用数据值来描述对象的状态;对象还有操作,用于改变对象的状态,对象及其操作就是对象的行为。对象实现了数据和操作的结合,使数据和操作封装在对象的统一体中。

在Java程序中可以使用new关键字来创建对象,如创建一个Book类的对象:

```
Book book= new Book();
```

这样Book类就被实例化为一个对象book。"new Book()"用于创建Book类的一个实例对象,"Book book"则是声明了一个Book类型的变量book。中间的等号用于将Book对象在内存中的地址赋值给变量book,这样变量book便持有了对象的引用。

3.4 封 装

封装(Encapsulation)是面向对象方法的重要原则,就是把对象的属性和操作(或服务)结合为一个独立的整体,并尽可能隐藏对象的内部实现细节。具备封装性的面向对象程序设计隐藏了某一方法的具体运行步骤,而是通过消息传递机制发送消息给它。

封装是通过限制只有特定类的对象可以访问这一特定类的成员,而它们通常利用接口实现消息的传入传出。通常来说,成员会根据它们的访问权限分为3种:公有成员(public)、私有成员(private)以及保护成员(protected)。例如,"狗"这个类有"吠叫()"的方法,这一方法定义了狗具体通过什么方法吠叫。但是,这条狗的朋友并不知道它到底是如何吠叫的。

对象的数据封装特性彻底消除了传统结构方法中数据与操作分离所带来的种种问题,提高了程序的可复用性和可维护性,降低了程序员保持数据与操作内容的负担。对象的数据封装特性还可以把对象的私有数据和公共数据分离开,保护了私有数据,减少了可能的模块间干扰,达到降低程序复杂性、提高可控性的目的。

通常情况下,我们把类中向外提供的接口声明为public,不想被外部看见的成员变量和内部方法等声明为private,私有的成员变量提供相应的get、set方法供外部调用。之所以将某些类内部的变量设为private属性,是因为需要对这些变量的访问作一定的约束,外部调用时,不会无意或恶意地将其内部变量的值进行修改。举个例子,小红和小黄是好朋友,小黄家里有一些好看的书籍,小红想去借来看,如果小红直接进入小黄家,那么,虽然可以得到书,但是,并没有得到小黄的允许,这是非法的行为,造成的结果也是不可预计的。所以,小黄把家门加了锁,小红想进小黄家拿书,那么必须找小黄拿钥匙,这样也就告知了小黄。小黄能够知晓其风险,这样出现未知情况的概率也就会下降。小黄家也安全了,小红也能够名正言顺地拿到他要的书籍。

3.3 节中的 Book 类被修改如下：

示例代码

```
public class Book {

    private String bookname;
    private String author;
    private int price;
    public Book(String bookname, String author, int bookPrice)
    {
        this.bookname = bookname;
        this.author = author;
        this.price = bookPrice;
    }
    public String getBookname() {
        return bookname;
    }
    public String getAuthor() {
        return author;
    }
    public int getPrice() {
        return price;
    }
    public void setBookname(String bookname) {
        this.bookname = bookname;
    }
    public void setAuthor(String author) {
        this.author = author;
    }

    public void setPrice(int price) {
        this.price = price;
    }
}
```

上述程序增加了访问器(get、set)方法，由于它们只返回实例域值，因此又称为域访问器。所以成员变量 bookname、author 和 price 可以被外部调用或修改。如果将 bookname、author 和 price 变量声明为 public 会不会更方便调用呢？关键在于 bookname、author 和 price 是只读域，一旦在构造器中设置完毕，就没有任何一个办法可以对它进行修改，这样一来确保 bookname、author 和 price 域不会受到外界的破坏。因此，我们通常使用的方法是声明一个 private 成员变量并提供它的 get 和 set 方法，这样做虽然比直接标记为共有变量复杂些，但好处也非常明显：一是可以改变内部实现，除了该类的方法之外，不会影响其他代码；二是更改器方法可以执行错误检查，然而直接对变量

赋值不会进行这些处理。

3.5 继 承

继承(Inheritance)是指在某种情况下,一个类会有“子类”,子类比原本的类(称为父类)要更加具体化。在 Java 中,继承的关键字是 extends,关键字 extends 表明正在构造的新类派生于一个已存在的类。这个已存在的类被称为超类(super class)、基类(base class)或父类(parent class);新类被称为子类(subclass)、派生类(derived class)或孩子类(child class),子类会自动拥有父类所有可继承的属性和方法。

然而,超类的有些方法对子类并不一定适用。为此,需要提供一个新的方法来覆盖(override)超类中的这个方法。如果在覆盖时,需要用到父类的方法,而恰巧子类中有与父类同名的方法,这时就要用到 super 关键字,super 关键字专门用于访问父类的成员。

这里用一个比较形象的例子来帮助大家理解继承的概念。例如,“狗”这个类可能会有它的子类“哈士奇”和“贵宾犬”等,在这种情况下,“小黑”可能就是“哈士奇”的一个实例,子类会继承父类的属性和行为,并且也可包含它们自己的。我们假设“狗”这个类有一个方法(行为)叫作“吠叫()”和一个属性叫作“毛皮颜色”,它的子类(前例中的哈士奇和贵宾犬)会继承这些成员,这意味着程序员只需要将相同的代码写一次。“哈士奇”这个类可以继承“毛皮颜色”这个属性,并指定其为黑色;而“贵宾犬”则可以继承“吠叫()”这个方法,并指定它的音调低于平常。子类也可以加入新的成员,例如,“贵宾犬”这个类可以加入一个方法叫作“颤抖()”。若用“哈士奇”这个类定义了一个实例“小黑”,那么小黑就不会颤抖,因为这个方法是属于“贵宾犬”的,而非“哈士奇”。

学过 C++的同学都知道 C++是支持多重继承的,而 Java 是不支持多继承的。多重继承存在会增加程序的复杂程度,使程序的编写和维护困难,容易出错,Java 的出现时间晚于 C++,因此它吸收了 C++的优点,规避了其缺点,不支持多继承恰恰就是 Java 编程语言的优势。虽然,Java 不支持多继承,但我们可以通过实现多个接口达到相同的目的,有关接口的概念笔者将在 3.7 节中详细讲解。

3.6 多 态

多态(Polymorphism)按字面的意思就是“多种状态”,它是面向对象程序设计(OOP)的一个重要特征。多态是同一个行为具有多个不同表现形式的能力。如果一个语言只支持类而不支持多态,只能说明它是基于对象的,而不是面向对象的。

在设计一个方法时,通常希望该方法具备一定的通用性。例如,动物都有叫这个行为(方法),由于每种动物的叫声是不同的,因此可以在方法中接收一个动物类型的参数,当传入猫类对象时就发出猫类的叫声,传入犬类对象时就发出犬类的叫声。在同一个方法中,这种由于参数类型不同而导致执行效果各异的现场就是多态。

在多态的学习中,涉及将子类对象当作父类类型使用的情况,例如下面两行代码:

```
Animal an1 = new Cat();            //将 Cat 对象当作 Animal 类型来使用
Animal an2 = new Dog();            //将 Dog 对象当作 Animal 类型来使用
```

将子类对象当作父类使用，即将子类的引用赋给父类对象，这又被称为指向子类的父类引用向上转型，需要注意的是，此时不能通过父类变量去调用子类中的某些方法（子类中存在而父类中不存在的方法），它只能访问父类中拥有的方法和属性。若子类重写了父类中的某些方法，在调用该些方法的时候，必定是使用子类中定义的这些方法。

对于面向对象而言，多态分为编译时多态和运行时多态。其中编辑时多态是静态的，主要是指方法的重载，它是根据参数列表的不同来区分不同的函数，通过编辑之后会变成两个不同的函数，在运行时谈不上多态；而运行时多态是动态的，它是通过动态绑定来实现的，也就是我们所说的多态性。简单地讲，重载就是同一类中的函数“见人说人话，见鬼说鬼话”的一种体现，重写则是子类在父类原有基础上的一种升级（如：父亲可以开汽车，儿子可以开飞机）。

上面提到的两个名词重载和重写，初学 Java 的同学往往会将它们搞混。笔者是这样区分的：重载是一个类中多态性的一种表现，具体指的是在同一个类中，具有相同的方法名，但是具有不同的参数列表，调用方法时，通过传递给它们的不同参数个数和参数类型来决定使用哪个方法。重写（覆盖）是子类对父类中某些方法的重新定义，具体指的是子类和父类具有相同的方法名、返回类型和参数列表，值得注意的是，重写时子类函数的访问权限不能少于父类。

总结一下，Java 实现多态有三个必要条件：继承、重写和向上转型。继承指的是在多态中必须存在有继承关系的子类和父类，重写是运行时多态的一种表现形式，向上转型是因为只有这样，该引用才能具备调用父类方法和子类方法的技能。因此，只有满足这三个条件，我们才能够在同一个继承结构中使用统一的逻辑使代码处理不同的对象，从而执行不同的行为。

下面笔者想讲一下 Java 中多态的两种实现方式：基于继承实现的多态和基于接口实现的多态。

（1）基于继承实现的多态。

主要表现在父类和继承该父类的一个或多个子类对某些方法的重写，多个子类对同一方法的重写可以表现出不同的行为。对于引用子类的父类类型，在处理该引用时，它适用于继承该父类的所有子类，子类对象的不同，对方法的实现也就不同，执行相同动作产生的行为也就不同。

如果父类是抽象类，那么子类必须要实现父类中所有的抽象方法，这样该父类所有的子类一定存在统一的对外接口，但其内部的具体实现可以各异。这样我们就可以使用顶层类提供的统一接口来处理该层次的方法。

（2）基于接口实现的多态。

继承是通过重写父类的同一方法的几个不同子类来体现的，也可以通过实现接口并覆盖接口中同一方法的几个不同的类来体现。在接口的多态中，指向接口的引用必须是指定实现了该接口的一个类的实例程序，在运行时，根据对象引用的实际类型来执行对应的方法。

Java 的继承都是单继承，只能为一组相关的类提供一致的服务接口。但是接口可以是多继承多实现，它能够利用一组相关或者不相关的接口进行组合与扩充，能够对外提供一致的服务接口。所以它相对于继承来说有更好的灵活性。

3.7　抽象类和接口

3.7.1　抽象类

在面向对象的领域一切都是对象，同时所有的对象都是通过类来描述的，但是并不是所有的类都是来描述对象的。如果一个类没有足够的信息来描述一个具体的对象，而需要其他具体的类来支撑它，那么这样的类我们称它为抽象类。例如前面提到的 Animal 类，Animal 具体长什么样子我们并不知道，它没有一个具体动物的概念，所以它就是一个抽象类，需要一个具体的动物，如狗、猫来对它进行特定的描述，我们才知道它长什么样。

抽象类除了不能实例化对象之外，类的其他功能依然存在，成员变量、成员方法和构造方法的访问方式和普通类一样。由于抽象类不能实例化对象，所以抽象类必须被继承，才能被使用。父类包含了子类集合的常见的方法，但是由于父类本身是抽象的，所以不能使用这些方法。

被关键字 abstract 修饰的方法称为抽象方法，当一个类中包含了抽象方法，该类必须使用 abstract 关键字来修饰，具体示例如下：

```
//定义抽象类 Animal
abstract class Animal {
        //定义抽象方法 shout()
        abstract int shout();
}
```

在定义抽象类时需要注意，包含抽象方法的类必须声明为抽象类，但抽象类可以不包含任何抽象方法，只需使用 abstract 关键字来修饰即可。另外，抽象类是不可以被实例化的，实例化的工作应该交由它的子类来完成，它只需要有一个引用即可。抽象方法必须由子类来进行重写，子类中的抽象方法不能与父类的抽象方法同名。

这里需要读者注意的是：abstract 不能与 final 并列修饰同一个类；abstract 不能与 private、static、final 或 native 并列修饰同一个方法。

3.7.2　接口

如果一个抽象类中的所有方法都是抽象的，则可以将这个类用另外一种方式来定义，即接口。接口是一系列方法的声明，是一些方法特征的集合，一个接口只有方法的特征没有方法的实现，因此这些方法可以在不同的地方被不同的类实现，而这些实现可以具有不同的行为（功能）。

在定义接口时，需要使用 interface 关键字来声明。在接口中定义的方法默认使用

"public abstract"来修饰，即抽象方法；在接口中定义的变量默认使用"public static final"来修饰，即全局变量。

接口是用来建立类与类之间的协议，它所提供的只是一种形式，而没有具体的实现。由于接口中的方法都是抽象方法，因此不能通过实例化对象的方式来调用接口中的方法。此时需要定义一个类，并使用 implements 关键字实现接口中所有的方法。接口是一种比抽象类更加抽象的"类"，这里的"类"加了引号表明接口本身就不是类，从不能实例化一个接口就可以看出。

接口是抽象类的延伸，Java 为了保证数据安全是不能多重继承的，也就是说继承只能存在一个父类。但是接口不同，一个类可以同时实现多个接口，不管这些接口之间有没有关系，所以接口弥补了抽象类不能多重继承的缺陷，但是推荐继承和接口共同使用，因为这样既可以保证数据安全性，又可以实现多重继承。

为了加深初学者对接口的认识，接下来对接口的特点进行归纳。

- 接口中的方法都是抽象的，不能实例化对象。
- 当一个类实现接口时，如果这个类是抽象类，则实现接口中的部分方法即可，否则需要实现接口中的所有方法。
- 一个类通过 implements 关键字实现接口时，可以实现多个接口，被实现的多个接口之间要用逗号隔开。
- 一个类在继承另一个类的同时还可以实现接口，此时，extends 关键字必须位于 implements 关键字之前。

3.8 访问控制

为了实现对类的封装和继承，Java 提供了访问控制机制。通过访问控制机制，类的设计者可以掩盖变量和函数来达到维护类自身状态的目的，而且还可以将另外一些需要暴露的变量和函数提供给别的类进行访问和修改。Java 一共提供了 4 种访问类型，它们分别是：公有型(public)、保护型(protected)、包访问(default)和私有型(private)。

这 4 种访问类型的控制级别由小到大依次为 private→default→protected→public，下面将通过一个表将这 4 种访问类型的访问级别更加直观地表示出来，如表 3-1 所示。

表 3-1 Java 关键字控制范围

访问控制修饰符	同一个类	同一个包	不同包的子类	不同包的非子类
private(私有的)	是	否	否	否
default(默认的)	是	是	否	否
protected(受保护的)	是	是	是	否
public(公共的)	是	是	是	是

3.9　异　　常

3.9.1　什么是异常

在程序运行过程中,可能会发生各种非正常状况,例如磁盘空间不足、网络连接中断、被装载的类不存在等。针对这种情况,Java 引入了异常,以异常类的形式对这些非正常情况进行封装,通过异常处理机制对程序运行时发生的各种问题进行处理。

要理解 Java 异常处理是如何工作的,你需要掌握以下三种类型的异常。

(1) 检查性异常:最具代表的检查性异常是用户错误或问题引起的异常,这是程序员无法预见的。例如要打开一个不存在文件时,一个异常就发生了,这些异常在编译时不能被简单地忽略。

(2) 运行时异常:运行时异常是可能被程序员避免的异常。与检查性异常相反,运行时异常可以在编译时被忽略。

(3) 错误:错误不是异常,而是脱离程序员控制的问题。错误在代码中通常被忽略。例如,当栈溢出时,一个错误就发生了,它们再编译也检查不到的。

Java 提供了大量的异常类,这些类都继承自 java.lang.Throwable 类。接下来将通过一张图来展示 Throwable 类的继承体系,如图 3-1 所示。

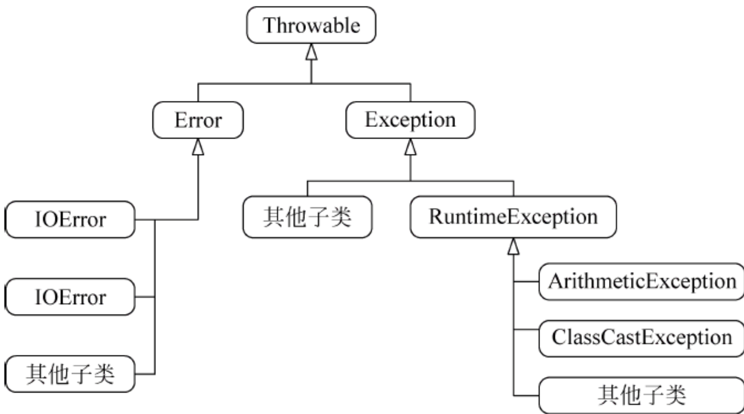


图 3-1　Throwable 体系架构图

通过图 3-1 可以看出,Throwable 有两个直接子类 Error 和 Exception,其中 Error 代表程序中产生的错误,它表示 Java 运行时产生的系统内部错误或资源耗尽的错误,是比较严重的,仅靠修改程序本身是不能恢复执行的。Exception 代表程序中产生的异常,它表示程序本身可以处理的错误,在开发 Java 程序中进行的异常处理,都是针对 Exception 类及其子类。

3.9.2　异常处理

在了解了什么是异常之后,下面我们将学习一些异常处理的方式。

(1) try/catch 语句。

最常用的就是使用 try/catch 语句捕获异常，将 try/catch 代码块放在异常可能发生的地方。try/catch 代码块中的代码称为保护代码，使用 try/catch 的语法如下：

示例代码

```
try
{
    //程序代码块
}catch(ExceptionType(Exception类及其子类) e){
    //catch块(对 ExceptionType的处理)
}
```

在 try 代码块中编写可能发生异常的 Java 语句，catch 代码块中编写针对异常进行处理的代码。当 try 代码块中的程序发生了异常，系统会将这个异常的信息封装成一个异常对象，并将这个对象传递给 catch 代码块。catch 代码块需要一个参数指明它所能够接收的异常类型，这个参数的类型必须是 Exception 类或其子类。此外，try 块还可以用多个 catch 块来捕获异常，这些异常是不同类型的，注意异常的变量名也必须是不同的。

需要注意的是，在 try 代码块中，发生异常语句后面的代码是不会被执行的。有时候，我们希望有些语句无论程序是否发生异常都要执行，这时可以在 try/catch 语句后，加一个 finally 代码块。

(2) finally 关键字。

无论程序发生异常还是使用 return 语句结束，finally 中的语句都会执行。因此，正是由于这种特殊性，在程序设计时，经常会在 try/catch 后使用 finally 代码块来完成必须做的事情，如释放系统资源。finally 的语法如下：

示例代码

```
try{
// ...
}catch(Exception1 e1){
    // ...
}catch(Exception2 e2){
    // ...
}finally{
    // ...
}
```

(3) throws/throw 语句。

前面我们讲了异常处理最常用的方式——捕获异常。由于 try 代码块中常常调用的是我们自己写的方法，因此该方法可能会发生异常。如果我们去调用一个别人写的方法，很难判断该方法是否会有异常。针对这种情况，Java 允许在方法的后面使用 throws 关键字对外声明该方法有可能发生的异常，这样调用者在调用方法时，就明确知道该方法有异

常,并且必须在程序中对异常进行处理。

使用 throws 或 throw 语句抛出异常,其中 throw 关键字是语句抛出异常,throws 关键字是声明一个异常(即通过方法抛出一个异常)。具体语法如下:

示例代码

```
public void func() throws Exception1, Exception2{
    try{
        //...
    } catch(Exception1 e1){
        throw e1;
    } catch(Exception2 e2){
        System.out.println("发生异常...");
    }
}
```

上述代码用两个 catch 块来捕获 try 块抛出的异常,如果产生 Exception1 异常,则捕获之后再抛出,由该方法的调用者去处理。如果产生 Exception2 异常,则由该方法自己处理异常(打印字符串),所以在函数声明处的 Exception2 就可以不用写了(写了也不会报错)。

最后,笔者想提醒大家,在学习异常时要注意以下事项。

- catch 不能独立于 try 存在。
- 在 try/catch 后面添加 finally 块并非强制性要求的。
- try 代码后不能既没有 catch 块也没有 finally 块。
- try/catch/finally 块之间不能添加任何代码。
- 要注意"自己的事情自己做",不要一味地抛出异常,最终全部都抛到 Exception 处,这样不利于程序员寻找错误。
- 同理,有多少异常就 catch 多少个异常,不建议只 catch 一个总的异常 Exception,这样不利于调试和纠错。

3.10 三个常见的关键字 static、final、this

3.10.1 static关键字

在 Java 中,定义了一个 static 关键字,它用于修饰类的成员,如成员变量、成员方法以及代码块等,被 static 修饰的成员具备一些特殊性。常见的 static 关键字的用法有以下三种。

1. 静态变量

在一个 Java 类中,可以使用 static 关键字来修饰成员变量,该变量被称作静态变量。静态变量和非静态变量的区别是:静态变量被所有的对象所共享,在内存中只有一个副本,当且仅当在类初次加载时它会被初始化,可以使用"类名.变量名"的形式来访问。而非静态变量是对象所拥有的,在创建对象的时候被初始化,存在多个副本,各个对象拥有

的副本互不影响。static 成员变量的初始化顺序按照定义的顺序进行初始化。

2. 静态方法

有时候,我们希望在不创建对象的情况下就可以调用某个方法,换句话说就是使该方法不必和对象绑在一起。要实现这样的效果,只需要在类中定义的方法前面加上 static 关键字即可,这种方法被称作静态方法。同静态变量一样,静态方法可以使用“类名.方法名”的方式来访问,也可以通过类的实例对象来访问。

由于静态方法不依赖于任何对象就可以进行访问,因此对于静态方法来说,是没有 this 的,因为它不依附于任何对象,既然都没有对象,就谈不上 this 了。并且由于这个特性,在静态方法中不能访问类的非静态成员变量和非静态成员方法,因为非静态成员方法/变量都是必须依赖具体的对象才能够被调用。但是要注意的是,虽然在静态方法中不能访问非静态成员方法和非静态成员变量,但是在非静态成员方法中是可以访问静态成员方法/变量的。

3. 静态代码块

在 Java 类中,使用一对花括号包围起来的若干行代码被称为一个代码块,用 static 关键字修饰的代码块称为静态代码块。当类被加载时,静态代码块会执行,由于类只加载一次,因此静态只执行一次。static 块可以置于类中的任何地方,类中可以有多个 static 块,系统会按照 static 块的顺序来执行每个 static 块,并且只会执行一次。

现在,我们回过头看一看 Java 的 main 函数。在 Java 的 main 函数之中,也有关键字 static,那么,为什么 main 函数中要使用 static 关键字呢?

static 关键字告知编译器 main 函数是一个静态函数。也就是说 main 函数中的代码是存储在静态存储区中的,静态方法在内存中的位置是固定的,即当定义了类以后这段代码就已经存在了。如果 main()方法没有使用 static 修饰符,那么编译不会出错,但是如果你试图执行该程序,将会报错,提示 main()方法不存在。因为包含 main()的类并没有实例化(即没有这个类的对象),所以其 main()方法也不会存在。而使用 static 修饰符则表示该方法是静态的,不需要实例化即可使用。

main()方法是一个程序的入口,如果写成非静态的,那么就必须实例化一个对象再来调用它,既然是入口这样肯定是不可以的;静态方法是属于类的,直接用类名就可以调用。相信看到这里,读者已经完全清楚了,main 函数为什么要使用 static 关键字修饰。下面笔者将继续给大家讲解 final 关键字。

3.10.2 final 关键字

final 在 Java 中是一个保留的关键字,可以用于修饰类、变量和方法,它有“无法改变的”或者“终态的”含义,因此被 final 修饰的类、变量和方法将具有以下特性。

- final 修饰的类不能被继承。
- final 修饰的方法不能被子类重写。
- final 修饰的变量(成员变量和局部变量)是常量,只能赋值一次。

接下来我们将对这些特性进行逐一地详解。

1. final 关键字修饰类

在 Java 中的类被 final 关键字修饰后，该类将不可以被继承，也就是不能够派生子类，因此 final 类的成员方法没有机会被覆盖，默认都是 final 的。在设计类的时候，如果这个类不需要有子类，类的实现细节不允许改变，并且确信这个类不会再被扩展，那么就设计为 final 类。

2. final 关键字修饰方法

当一个类的方法被 final 关键字修饰后，这个类的子类将不能重写该方法。如果你认为一个方法的功能已经足够完整了，子类中不需要改变的话，你可以声明此方法为 final。final 方法比非 final 方法要快，因为在编译的时候已经静态绑定了，不需要在运行时再动态绑定。

3. final 关键字修饰变量

Java 中被 final 修饰的变量为常量，它只能被赋值一次，也就是说 final 修饰的变量一旦被赋值，其值不能改变。final 变量经常和 static 关键字一起使用，作为常量。

3.10.3 this 关键字

在 3.3 节的示例代码中，出现了形如“this.bookname＝bookname；”这样的代码，这段代码中涉及 this 关键字。在 Java 中，为了解决成员变量和局部变量名称冲突的问题，提供了一个关键字 this，用于在方法中访问对象的其他成员。下面将详细地讲解 this 关键字在程序中的三种常见用法。

(1) 通过 this 关键字访问一个类的成员变量，解决与局部变量名称冲突的问题。

(2) 在成员方法中，通过 this 关键字调用其他的成员方法。

(3) 在构造方法中，通过使用“this([参数 1，参数 2……])”的形式来调用其他的构造方法。

在使用 this 调用类的构造方法时，应注意以下几点。

- 只能在构造方法中使用 this 调用其他的构造方法，不能在成员方法中使用。
- 在构造方法中，使用 this 关键字调用构造方法的语句必须位于第一行，且只能出现一次。
- 不能在一个类的两个构造方法中使用 this 互相调用。

3.11 图书管理系统 V3.0

3.11.1 运行效果图

图 3-2 展示了图书管理系统 V3.0 的运行效果。

3.11.2 类结构示意图

图 3-3 和图 3-4 分别展示了类的结构图和类的 MVC 三层结构图。

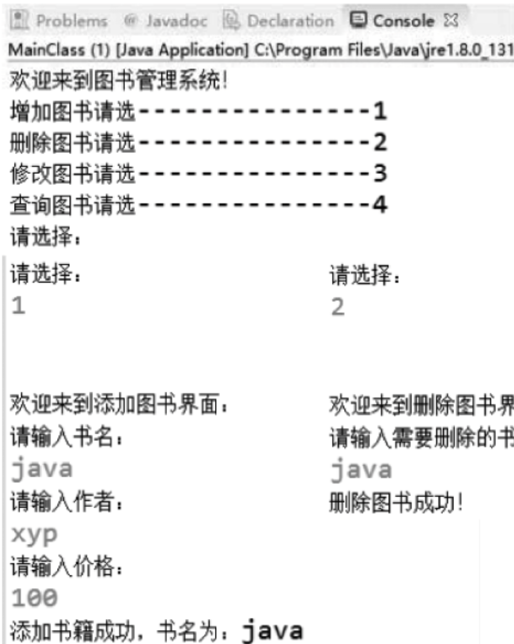


图 3-2 运行效果图

(1) 类结构

(2) 类 MVC 三层结构

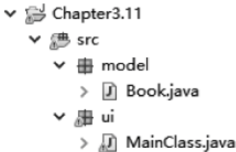


图 3-3 类结构图



图 3-4 类 MVC 三层结构图

3.11.3 代码实现

如运行效果图所示，笔者只实现了图书管理系统的增加和删除功能，修改和查找留给读者自行完成，具体代码实现如下：

MainClass.java

```
package ui;

import java.util.Scanner;
import model.Book;

public class MainClass {
    private int bookNum = 0;
    private static final int MAXNUM = 100;
    private Book[] bookList= new Book[MAXNUM];
    public MainClass() {
    Scanner input = new Scanner(System.in);
        while (true)
        {
            System.out.println("欢迎来到图书管理系统!");
            System.out.println("增加图书请选---------------1");
            System.out.println("删除图书请选---------------2");
            System.out.println("修改图书请选---------------3");
            System.out.println("查询图书请选---------------4");
            System.out.println("请选择:");
            int chioce = input.nextInt();
            switch (chioce){
            case 1:
                addBook();
                break;
            case 2:
                deleteBook();
                break;
            case 3:
                changeBook();
                break;
            case 4:
                searchBook();
                break;
            default:
                break;
            }
        }
    }
    public void addBook()
    {
        Scanner input= new Scanner(System.in);

        System.out.println("\n\n欢迎来到添加图书界面:");
```

```

        System.out.println("请输入书名:");
        String bookName = input.nextLine();

        System.out.println("请输入作者:");
        String bookAuthor = input.nextLine();

        System.out.println("请输入价格:");
        int bookPrice = input.nextInt();
        Book book = new Book(bookName, bookAuthor, bookPrice);
        bookList[bookNum++] = book;

        System.out.println("添加书籍成功,书名为:"+bookList[0].bookname);

    }
    public void deleteBook()
    {
        Scanner input = new Scanner(System.in);
        System.out.println("\n\n欢迎来到删除图书界面:");

        System.out.println("请输入需要删除的书名:");
        String bookname = input.nextLine();
        for(int i=0;i<bookNum;i++)
        {
            if(bookList[i].bookname.equals(bookname))
            {
                bookList[i]=bookList[i+1];
                System.out.println("删除图书成功!");

            }
        }
    }
    public void changeBook()
    {

    }
    public void searchBook()
    {

    }
}
```

Book.java

```
package model;

```

```
public class Book {

    public String bookname;

    public String author;

    public int price;

    public Book(String bookname, String author, int price)
    {
        this.bookname = bookname;
        this.author = author;
        this.price = price;
    }

}
```

第 4 章

集　合

从本章开始，我们将开始对 Java 中较为重要的数据结构进行介绍，我们会详细讲解数据在内存中保存的形式，以及它们的优缺点等。

4.1　本章任务

理论任务：在 Java 中，集合是 Java 对数据结构的表现，本章将介绍 Java 集合的概念，集合的总体框架的概念以及如何使用迭代器和比较器两个工具类。

实践任务：在第 3 章的学习中，我们了解了如何定义 Book 类以及如何将书籍的信息存入 Book 类。并使用 Book[]数组作为容器对书籍进行存储。但是，使用数组对书籍进行存储，存在一个大小受限的问题。因此，本章我们将对 Book 对象的存储进行一次“升级”，在不影响系统整个功能的前提下，对容器进行改造，使用的新容器就是集合。

4.2　集合——数据结构 Java 实现

在第 3 章中通过数组来保存书籍对象，但是随着书籍的增加与删除，这时书籍的数量是很难确定的。为了保存这些数目不确定的对象，JDK 中提供了一系列特殊的类，这些类可以存储任意类型的对象，并且长度可变，统称为集合。

Java 集合(类)是 Java 提供的工具包，包含了常用的数据结构：集合、链表、队列、栈、数组、映射等。所有 Java 集合类都位于 java.util 包中，在使用时一定要注意导包的问题，否则会出现异常。

集合与数组的不同点在于数组的长度固定，不适合在对象数量未知的情况下使用，且数组只能通过下标访问元素，下标类型只能是数字型；而集合可以存储任意类型的对象，并且长度可变，且有的集合可以通过任意类型查找所映射的具体对象。与 Java 数组不同，Java 集合中不能存放基本数据类型，只能存放对象的引用。

4.3　Java 集合的整体框架

Java 集合类库构成了集合类的框架。它为集合的实现者定义了大量的接口和抽象类，并对其中的某些机制给予了描述。换句话说，集合框架就是为表示和操作集合而规定的一种统一的标准的体系结构。

Java 集合按照其存储结构可以分为两大类，即单列集合 Collection 和双列集合 Map，这两种集合的特点具体如下。

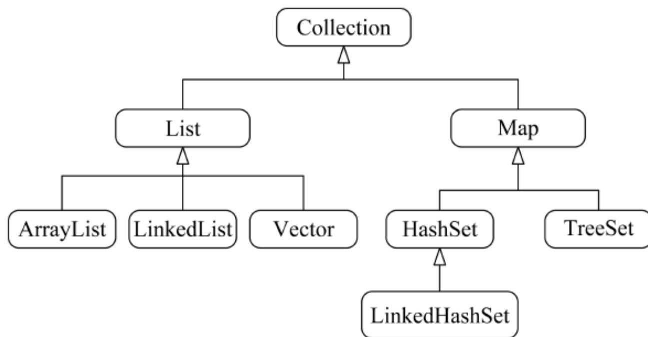
- Collection：单列集合类的根接口，用于存储一系列符合某种规则的元素，它有两个重要的子接口，分别是 List 和 Set。其中，List 的特点是元素有序，元素可重复；Set 的特点是元素无序，元素不可重复。List 接口的主要实现类有 ArrayList 和 LinkedList；Set 接口的主要实现类有 HashSet 和 TreeSet。
- Map：双列集合类的根接口，用于存储具有键(Key)、值(Value)映射关系的元素，每一个元素都包含一对键值，在使用 Map 集合时可以通过指定的 Key 找到对应的 Value。Map 接口的主要实现类有 HashMap 和 TreeMap。

为便于读者进行系统的学习，接下来通过三张图来描述整个集合类的继承体系，如图 4-1 所示。

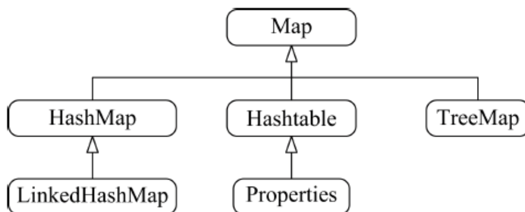
(1) 单列集合，如图 4-1(a)所示。

(2) 双列集合，如图 4-1(b)所示。

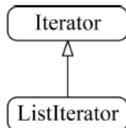
(3) 迭代器，如图 4-1(c)所示。



(a) Collection集合架构图



(b) Map集合架构图



(c) Iterator迭代器架构图

图 4-1 集合体系架构图

图 4-1 列出了程序中常用的一些集合类，后面的小节将会对图中所列举的部分集合类进行详细讲解。

4.4 Collection 接口

Collection 是所有单列集合的父接口，因此在 Collection 中定义了单列集合（List 和 Set）通用的一些方法，这些方法可用于操作所有的单列集合，如表 4-1 所示。

表 4-1 Collection 接口的方法

方法声明	功能描述
boolean add(Object o)	向集合中添加一个元素
boolean addAll(Collection c)	将指定 Collection 中所有元素添加到该集合中
void clear()	删除该集合中的所有元素
boolean remove(Object o)	删除该集合中指定的元素
boolean removeAll(Collection c)	删除指定集合中的所有元素
boolean isEmpty()	判断该元素是否为空
boolean contains(Object o)	判断该集合中是否包含某个元素
boolean containsAll(Collection c)	判断该集合中是否包含指定集合中的所有元素
Iterator iterator()	返回在该集合的元素上进行迭代的迭代器（Iterator），用于遍历该集合所有元素
int size()	获取该集合元素个数

表 4-1 所列举的方法，都来自于 Java API 文档，建议初学者通过查询 API 文档来学习这些方法的具体用法。

4.5 List 接口

4.5.1 List 接口简介

List 接口是包含有序元素的一种 Collection 子接口，其中元素必须按序存放，元素的存入顺序和取出顺序一致。List 作为 Collection 集合的子接口，不但继承了 Collection 接口中的全部方法，而且还增加了一些根据元素索引来操作集合的特有方法。List 接口的主要方法如表 4-2 所示。

表 4-2 List 接口的主要方法

方法声明	功能描述
void add(int index,Object element)	将元素 element 插入在 List 集合的 index 处
boolean addAll(int index,Collection c)	将集合 c 所包含的所有元素插入在 List 集合的 index 处
ListIterator listIterator()	返回一个 ListIterator
ListIterator listIterator(int index)	返回指定的 ListIterator
Object get(int index)	返回集合索引 index 处的元素

续表

方法声明	功能描述
Object remove(int index)	删除 index 索引处的元素
Object set(int index,Object element)	将索引 index 处元素替换成 element 对象,并将替换后的元素返回
int indexOf(Object o)	返回对象 o 在 List 集合中出现的位置索引
int lastIndexOf(Object o)	返回对象 o 在 List 集合中最后一次出现的位置索引
List subList(int fromIndex,int toIndex)	返回从索引 fromIndex(包括)到 toIndex(不包括)处所有元素集合组成的子集合

List 接口派生出了 ArrayList、LinkedList、Vector、Stack 几个子类。本节介绍 ArrayList 和 LinkedList 类的用法,Vector 和 Stack 由于用得比较少,不在本书介绍的范围内。同时,本节还将给大家介绍一下利用迭代器(Iterator)实现元素遍历的方法。

4.5.2 ArrayList 集合

ArrayList 是 List 接口的一个可变长数组的实现,即一个 ArrayList 类对象可以动态改变大小。每个 ArrayList 类对象都有一个容量(Capacity),用于存储元素数组的大小。容量可随着不断添加新元素和删除旧元素而自动变大、变小。集合中允许存储 null 值。ArrayList 类的随机访问速度快,但是向表中插入和删除比较慢。

ArrayList 常用的构造函数如下。

- ArrayList():构建一个空的 ArrayList 对象。
- ArrayList(Collection c):构建一个 ArrayList 对象,并且将集合 c 中所有元素添加进去。
- ArrayList(int initialCapacity):构建一个拥有特定容量的空的 ArrayList 对象。

示例代码

```
import java.util.ArrayList;

public class ArrayListExample {

    public static void main(String[] args) {

        ArrayList list=new ArrayList();

        list.add("book1");
        list.add("book2");
        list.add("book3");
        System.out.println("集合长度:"+list.size());
        System.out.println("第 2个元素是:"+list.get(1));
    }

}
```

在上例中，首先调用add(Object o)方法向ArrayList集合添加了3个元素，然后调用size()方法获取集合中元素个数，最后通过调用get(int index)方法取出指定索引位置的元素。需要注意的是，集合和数组一样，索引的取值范围是从0开始的，最后一个索引是size-1。

4.5.3 LinkedList集合

ArrayList集合在查询元素时速度很快，但是增删元素时效率较低，为了克服这种局限性，可以使用List接口的另一个实现类LinkedList。该集合提供了使用双向链表实现数据存储的方法，可按序号检索数据，并能够向前或向后遍历。由于插入数据时只需要记录元素的前后项即可，所以插入速度较快，因此适合于在链表中间需要频繁进行插入和删除的操作。LinkedList类中的常用方法如表4-3所示。

表4-3 LinkedList中定义的方法

方法声明	功能描述
void add(int index,Object o)	将对象o添加到链表中由index指定位置
void addFirst(Object o)	将指定元素插入此链表的开头
void addLast(Object o)	将指定元素添加到此链表的结尾
Object getFirst()	返回此链表第一个元素
Object getLast()	返回此链表最后一个元素
Object removeFirst()	移除并返回此链表的第一个元素
Object removeLast()	移除并返回此链表的最后一个元素

LinkedList的构造方法如下。

- LinkedList()：创建一个空链表。
- LinkedList(Collection c)：创建一个以集合c中元素为初始值的链表。

LinkedList的很多成员方法与ArrayList相似，两者的本质区别是一个使用链表结构，另一个使用顺序结构，因此，它也可以使用ArrayList类提供的方法进行列表的操作。

4.6 Set接口

4.6.1 Set接口简介

Set接口和List接口一样，同样继承自Collection接口，它与Collection接口中的方法基本一致，并没有对Collection接口进行功能上的扩充，只是比Collection接口更加严格了。与List接口不同的是，Set接口中元素无序，并且都会以某种规则保证存入的元素不出现重复。

Set接口派生了一个SortedSet接口和一个抽象类AbstractSet。SortedSet接口用来描述有序的元素集合，TreeSet实现了这个接口，它将放入其中的元素按序存放，要求其

中的对象是可排序的。抽象类 AbstractSet 实现了部分 Collection 接口，并有一个子类 HashSet，它以散列方式表示集合内容。

HashSet 是根据对象的哈希值来确定元素在集合中的存储位置，因此具有良好的存取和查找性能；TreeSet 则是以二叉树的方式来存储元素，它可以实现对集合中的元素进行排序。接下来，笔者将围绕 Set 的两个实现类详细地进行讲解。

4.6.2 HashSet 集合

HashSet 类是实现了 Set 接口的标准类，它创建了一个使用哈希表存储的集合，能快速定位一个元素，从而可以优化查询速度，特别是在查找大集合时 HashSet 类比较有用。当向 HashSet 集合中添加一个对象时，首先会调用该对象的 hashCode()方法来确定元素的存储位置，然后再调用对象的 equals()方法来确保该位置没有重复元素。

HashSet 的构造函数如下。

- HashSet()：创建一个空的哈希集。
- HashSet(Collection c)：创建一个哈希集，并且将集合 c 中所有元素添加进去。
- HashSet(int initialCapacity)：创建一个拥有特定容量的空哈希集。
- HashSet(int initialCapacity，float loadFactor)：创建一个拥有特定容量和加载因子的空哈希集。loadFactor 是 0～1.0 之间的一个数，默认为 0.75。加载因子定义了哈希集充满什么程度时就要增加容量。

4.6.3 TreeSet 集合

TreeSet 是 Set 接口的另一个实现类，它内部采用自平衡的排序二叉树来存储元素，这样的结构可以保证 TreeSet 集合中没有重复的元素，并且可以对元素进行排序。

当把一个对象加入 TreeSet 集合时，TreeSet 调用该对象的 compareTo(Object obj)方法与容器中的其他对象比较大小，然后根据自平衡的排序二叉树结构找到它的存储位置，再把这两个对象通过 compareTo(Object obj)方法进行比较，若大小相等，则新对象将无法添加到 TreeSet 集合中。

向 TreeSet 集合中依次存入元素如下：首先将第 1 个存入的放在二叉树的最顶端，之后存入的元素与第 1 个元素比较，小于就将该元素放在左子树上，大于就将该元素放在右子树上，依次类推，按照左子树元素小于右子树元素的顺序进行排序。当所存元素与二叉树中已经存的元素相等时，TreeSet 会把重复的元素去掉(有关二叉树的概念和特点建议读者自行查阅数据结构相关书籍了解)。

4.7 Map 接口

4.7.1 Map 接口简介

在应用程序中，如果想存储具有对应关系的数据，则需要使用 Map(映射)接口。Map 接口是一种双列集合，它与 List 或 Set 有明显的区别，Map 中每项都是成对出现的，它提

供了一组键值的映射。其中存储的每个对象都有一个相应的关键字(Key),关键字决定了对象在 Map 中的存储位置。关键字应该是唯一的,每个 key 只能映射一个值对象 value。Map 相关方法如表 4-4 所示。

表 4-4　Map 接口的主要方法

方法声明	功能描述
void put(Object key,Object value)	将指定的值与此映射中的指定键关联(可选操作)
Object get(Object key)	返回指定键所映射的值;如果此映射不包含该键的映射关系,则返回 null
boolean containsKey(Object key)	如果此映射中包含指定键的映射关系,则返回 true
boolean containsValue(Object value)	如果此映射将一个或多个键映射到指定值,则返回 true
Set keySet()	返回此映射中包含的键的 Set 视图
Collection <V> values()	返回此映射中包含的值的 Collection 视图
Set <Map.Entry<K,V>> entrySet()	返回此映射中包含的映射关系的 Set 视图

表 4-4 列出了一系列方法用于操作 Map,其中,put(Object key,Object value)和 get(Object key)方法分别用于向 Map 中存入和取出元素;containsKey(Object key)和 containsValue(Object value)方法分别用于判断 Map 中是否包含某个指定的键或值;keySet()和 values()方法分别用于获取 Map 中所有的键和值。

Map 并不是一个真正意义上的集合,但是这个接口提供了三种"集合视角",使得可以像操作集合一样操作它们,具体如下。

- 把 Map 的内容看作 key 的集合。
- 把 Map 的内容看作 value 的集合。
- 把 Map 的内容看作 key-value 映射的集合。

Map 接口提供了大量的实现类,常用的有 HashMap 和 TreeMap,接下来针对这两个类进行详细的讲解。

4.7.2 HashMap 集合

HashMap 集合是 Map 接口的一个实现类,它用于存储键值映射关系,但必须保证不出现重复的键。在 Map 中插入、删除和定位元素,HashMap 是最好的选择。

HashMap 的构造方法如下。

- HashMap():创建一个空的 HashMap 集合。
- HashMap(Map t):创建一个哈希集,并且将 t 中所有元素添加进去。
- HashMap(int initialCapacity):创建一个拥有特定容量的空 HashMap 集合。
- HashMap(int initialCapacity,float loadFactor):创建一个拥有特定容量和加载因子的空 HashMap。

表 4-5 列出了 HashMap 的主要成员方法。

表 4-5　HashMap 类的主要成员方法

方 法 声 明	功 能 描 述
Object put(Object key,Object value)	用键值 key 存储对象 value
void putAll(Map map)	将 map 中的所有键值/对象传递给当前的散列表
Object get(Object key)	返回键值 key 所对应的对象
remove(Object key)	删除 key 键值所对应的对象
Set keySet()	返回一个 Set 对象,其内容为所有的键值
Set entrySet()	返回一个 Set 对象,其内容为所有的键值/对象对
Collection values()	返回一个 Collection 对象,其内容为散列表中存储的所有对象
Object getKey()	返回对象的键值
Object getValue()	返回所对应的对象
void setValue(Object new)	将对象设置为 new

4.7.3　TreeMap 集合

Map 接口还有一个常用的实现类 TreeMap。TreeMap 集合是用来存储键值映射关系的,其中不允许出现重复的键。在 TreeMap 中是通过二叉树的原理来保证键的唯一性,这与 TreeSet 集合存储原理一样,因此 TreeMap 中所有的键是按照某种顺序排列的。

接下来通过一个例子来了解一下 TreeMap 的具体用法。

示例代码

```
import java.util.*;
public class TreeMapExample {
    public static void main(String[] args) {
        TreeMap tm=new TreeMap ();
        tm.put("1","book1");
        tm.put("2","book2");
        tm.put("3","book3");
        Set keySet=tm.keySet();
        Iterator iterator=keySet.iterator();
        while (iterator.hasNext()) {
            Object key=iterator.next();
            Object value=tm.get(key);
            System.out.println(key+":"+value);
        }
    }
}
```

在上面的例子中,使用 put()方法将三本书籍的信息存入 TreeMap 集合,其中书籍的编号作为键,书名作为值,然后对书籍信息进行遍历。注意,取出的元素是按照书籍编号的自然顺序进行排序的,这是因为书籍编号是 String 类型,String 类实现了 Comparable 接口,因此默认会按照自然顺序进行排序。

截至这里,笔者已经将本章所需要介绍的集合基本介绍完了,为了让大家对于本章所

学集合有一个整体的认识，笔者将本章介绍的各个集合进行了对比，具体如表 4-6 所示。

表 4-6　各个不同集合类间的对比表

集合			是否有序	是否允许元素重复
Collection	List	ArrayList	是	是
		LinkedList		
	Set	HashSet	否	否
		TreeSet	是(用二叉排序树)	
Map		HashMap	否	使用 key-value 来映射和存储数据，key 必须唯一，value 可以重复
		TreeMap	是(用二叉排序树)	

4.8　常用的三个工具：Iterator 接口、Collections 类、Arrays 类

4.8.1　Iterator 接口

在程序的开发中，经常需要遍历集合中的所有元素，针对这种需求，JDK 专门提供了一个接口 Iterator。Iterator 接口也是 Java 集合框架中的一员，但它与 Collection 和 Map 接口不同，Collection 和 Map 接口主要用于存储元素，而 Iterator 主要用于迭代访问(遍历)Collection 中的元素，因此 Iterator 对象也被称为迭代器。Iterator 接口的主要方法如表 4-7 所示。

表 4-7　Iterator 接口的主要方法

方法声明	功能描述
boolean hasNext()	判断是否还有其他元素
Object next()	获取下一个元素
void remove()	删除最后一次调用 next 方法返回的元素
Set keySet()	返回 Set 类型的接口

接下来通过一个例子来学习如何使用 Iterator 迭代集合中的元素。

示例代码

```
import java.util.ArrayList;
import java.util.Iterator;
public class IteratorExample {

    public static void main(String[] args) {
        ArrayList list=new ArrayList();
        list.add("book1");
        list.add("book2");
        list.add("book3");
        Iterator iterator=list.iterator();
        while (iterator.hasNext()) {
```

```
            Object object=iterator.next();
            System.out.println(object);
        }
    }
}
```

从上面的例子可以看出,当遍历元素时,首先通过调用 ArrayList 集合的 iterator()方法获得迭代器对象,然后使用 hasNext()方法判断集合中是否存在下一个元素,如果存在,则调用 next()方法将元素取出,否则就说明已经达到集合末尾,停止遍历元素。

Iterator 迭代器对象在遍历集合时,内部采用指针的方式来跟踪集合中的元素。在调用 Iterator 的 next()方法之前,迭代器的索引位于第一个元素之前,不指向任何元素,当第一次调用迭代器的 next()方法后,迭代器的索引会向后移动一位,指向第一个元素并将该元素返回,当再次调用 next()方法时,迭代器的索引会指向第二个元素并将该元素返回,依次类推,直到 hasNext()方法返回 false,表示到达了结合的末尾,终止对元素的遍历。

需要特别注意的是,当通过迭代器获取 ArrayList 集合中的元素时,都会将这些元素当作 Object 类型来看待,如果想得到特定类型的元素,则需要进行强制类型转换。

Collection 接口的 iterator()和 toArray()方法都用于获得集合中的所有元素,前者返回一个 Iterator 对象,后者返回一个包含集合中所有元素的数组。

4.8.2 Collections 类

在程序中,针对集合的操作非常频繁,例如将集合中的元素排序、从集合中查找某个元素等。针对这些常见操作,JDK 提供了一个工具类专门用来操作集合,这个类就是 Collections,它位于 java.util 包中。

Collections 是对集合框架的一个工具类,它里边的方法都是静态的,不需要创建对象,并未封装特有数据。在 Collections 工具类中大部分方法是用于对 List 集合进行操作的,如比较、二分查找、随机排序等。

1. 排序操作

Collections 类中提供了一系列方法用于对 List 集合进行排序,具体如表 4-8 所示。

表 4-8 Collections 常用 List 方法

方法声明	功能描述
static <T> boolean addAll(Collection <? super T> c,T…elements)	将所有指定元素添加到指定的 collection 中
static void reverse(List list)	反转指定 List 集合中元素顺序
static void shuffle(List list)	对 List 集合中的元素进行随机排序
static void sort(List list)	根据元素的自然顺序对 List 集合中的元素进行排序
static void swap(List list,int i,int j)	将指定 List 集合中 i 处元素和 j 处元素进行交换

2. 替换操作

Collections 类还提供了一些常用方法用于查找、替换集合中的元素,如表 4-9 所示。

表 4-9 Collections 常用其他方法

方法声明	功能描述
static int binarySearch(List list,Object key)	使用二分法搜索指定对象在 List 集合中的索引,查找的 List 集合中的元素必须是有序的
static Object max(Collection col)	根据元素的自然顺序,返回给定集合中最大的元素
static Object min(Collection col)	根据元素的自然顺序,返回给定集合中最小的元素
static boolean replaceAll(List list, Object oldVal,Object newVal)	用一个新的 newVal 替换 List 集合中所有的旧值 oldVal

3. Collection 和 Collections 的区别

Collection 是单列集合的顶层接口,有子接口 List 和 Set; Collections 是针对集合操作的工具类,有对集合进行排序和二分查找的方法。

4.8.3 数组工具类 Arrays

java.util 包中还提供了一个专门用于操作数组的工具类——Arrays。Arrays 工具类里边的方法也全是静态的,不需要创建对象。读者可能要问,把数组变成 List 集合有什么好处呢?是的,把数组变成 List 集合可以使用集合的思想和方法来操作数组中的元素,如:contains、get、indexOf、subList 等方法。

在前面学习数组的时候,要想对数组进行排序就需要自定义一个排序方法,其实也可以使用 Arrays 工具类的静态方法 sort()来实现这个功能。除 sort()方法以外,Arrays 类还提供了很多方法,这里就不一一列举了,下面介绍几个最常用的方法。

- public static String toString(int[] a):返回数组的字符串形式。
- public static void sort(int[] a):排序。
- public static intbinarySearch(int[]a,int key):二分查找。
- asList(T... a):返回一个受指定数组支持的固定大小的 list 集合。

示例代码

```
import java.util.Arrays;

public class ArraysDemo {
    public static void main(String[] args) {
        //定义一个数组
        int[] arr = { 24, 69, 80, 57, 13 };
        //把数组转成字符串
        System.out.println("排序前:" + Arrays.toString(arr));
        //对数组进行排序
        Arrays.sort(arr);
        System.out.println("排序后:" + Arrays.toString(arr));
        //二分查找
        System.out.println("binarySearch:" + Arrays.binarySearch(arr, 57));
        System.out.println("binarySearch:" + Arrays.binarySearch(arr, 577));
```

```
        }
}
```

4.9　图书管理系统 V4.0

4.9.1　运行效果图

本章的运行效果图与图书管理系统 V3.0 的运行效果图一样，因此这里就不再重复粘贴。正如本章实践任务所提，本章只是对图书管理系统存储图书的容器进行了一个升级，运行效果与之前完全一致。

4.9.2　类结构示意图

同运行效果图一样，本章的类结构示意图也与图书管理系统 V3.0 的类结构示意图一样，如果读者不太清楚，可以返回第 3 章查看，本章只是将 MainClass.java 类的第 9 行的数组替换成了集合。

4.9.3　代码实现

如运行效果图所示，笔者只实现了图书管理系统的增加和删除功能，修改和查找留给读者自行完成，具体代码实现如下。

MainClass.java

```
1   package ui;
2
3   import java.util.Scanner;
4   import java.util.Vector;
5
6   import model.Book;
7
8   public class MainClass {
9       private Vector<Book> bookList = new Vector<>();
10
11      public MainClass() {
12      Scanner input = new Scanner(System.in);
13          while (true)
14          {
15              System.out.println("欢迎来到图书管理系统!");
16              System.out.println("增加图书请选---------------1");
17              System.out.println("删除图书请选---------------2");
18              System.out.println("修改图书请选---------------3");
19              System.out.println("查询图书请选---------------4");
20              System.out.println("请选择:");
21              int chioce = input.nextInt();
```

```
            switch (chioce){
            case 1:
                addBook();
                break;
            case 2:
                deleteBook();
                break;
            case 3:
                changeBook();
                break;
            case 4:
                searchBook();
                break;
            default:
                break;
            }
        }
    }

    public static void main(String[] args) {
        new MainClass();
    }
    public void addBook()
    {
        Scanner input = new Scanner(System.in);

        System.out.println("\n\n欢迎来到添加图书界面:");

        System.out.println("请输入书名:");
        String bookName = input.nextLine();

        System.out.println("请输入作者:");
        String bookAuthor = input.nextLine();

        System.out.println("请输入价格:");
        int bookPrice = input.nextInt();

        Book book = new Book(bookName, bookAuthor, bookPrice);
        bookList.add(book);
        System.out.println("添加书籍成功,书名为:"+bookList.get(0).bookname);

    }
    public void deleteBook()
    {
```

```
            Scanner input = new Scanner(System.in);
            System.out.println("\n\n欢迎来到删除图书界面：");
            System.out.println("请输入需要删除的书名：");
            String bookname = input.nextLine();
            for(int i=0;i<bookList.size();i++)
            {
                if(bookList.get(i).bookname.equals(bookname))
                {
                    bookList.remove(i);
                    System.out.println("删除图书成功！");
                }
            }
        }
        public void changeBook()
        {

        }
        public void searchBook()
        {

        }
}
```

Book.java

```
package model;

public class Book {

        public String bookname;

        public String author;

        public int price;

        public Book(String bookname, String author, int price)
        {
            this.bookname = bookname;
            this.author = author;
            this.price = price;
        }

}
```

第 5 章

数 据 存 储

本章中,我们将了解如何进行读取和存储数据、数据存储的不同形式及其所涉及的相关知识,由此我们便能够分辨出它们的优缺点。

5.1 本 章 任 务

理论任务:了解 I/O 的基本概念,对于文件系统有基本认识,学会使用 JDBC 连接数据库,并对数据库中的数据进行存储和读取,同时学习 MVC 设计模式,将编写的程序代码按照 MVC 设计模式进行重构。

实践任务:在之前的章节中,booklist 中的内容都是存储在内存中的,不能保证数据的长期存储。在这一章中,我们将会把数据存储在硬盘中,通过文件或者数据库的形式对数据进行长期保存,保证数据不丢失。同时,我们将讲解 MVC 的设计模式,并对之前的代码进行重构,将显示内容、数据操作类、控制类进行分类和归类,降低程序耦合度,以便提高开发效率、方便代码维护。

5.2 IO

IO(输入输出)在 Java 及众多编程语言中都是极其重要的一部分,在 Java 中,输入输出主要采用数据流的方式实现,在本节中,我们将讲解基本的输入输出方法。

5.2.1 基本 IO

正如 2.7 节中提到的,按照标准的 I/O 模型,Java 提供了 System. in、System. out 进行输入输出操作,在使用过程中,因为 System. out 被包装,我们可以直接使用,但是 System. in 是一个没有被包装过的 InputStream,所以在使用过程中,我们需要自己包装 System. in。并且在其使用过程中,通常我们使用 readline()一次一行读取数据,但是在实际操作中,除了对数据进行一行一行的读取,我们还可能对数据进行数组的存储或者字符的截取。这时,我们会发现,针对不知道长度或者大小的输入,我们的操作会变得困难而复杂,所以在下节中,我们将会讲解更方便的 IO 操作。

5.2.2 更好用的 IO

Java 的输入输出功能都是基于类库 java.io 包来实现,java.io 库提供了全面的 IO 接口,并且 Java 中 IO 是以流为基础的。流是什么呢? 流是一组有顺序的,有起点和终点的字节集合,当程序需要读取数据的时候,就会开启一个通向数据源的流,Java 将来自不同源和目标的数据统一抽象为数据流。

在 Java 的 IO 操作中,对流的操作分为读和写两种。根据流的运动方向,我们将流分为输入流和输出流,这里的输入输出都是以计算机内存为参照物的,所以,从键盘等外设流入计算机内存的数据序列称为输入流,反之,从计算机内存流出的数据序列称为输出流。

Java 按照流中元素的基本类型,将数据流分为字节流和字符流。以字节为单位传输数据的流称为字节流;以字符为单位传输数据的流称为字符流。根据功能的不同,又将流分为节点流和处理流。直接从数据源读写数据的流为节点流;从其他的流上进行数据处理的流为处理流。Java 中所有的流类型分别继承四种抽象流类,如表 5-1 所示。

表 5-1 四种抽象输入输出流

	字 节 流	字 符 流
输入流	InputStream	Reader
输出流	OutputStream	Write

在本节中我们将根据流的分类从输入和输出方面讲解更好用的 IO。

1. 输入流

在 Java 中,把能够读取一个字节序列的对象称为字节输入流,把能够写一个字节的对象称为字节输出流。

输入流又分为字节输入流和字符输入流,这两个类下又分为多个类,其中,字节输入流依靠 InputStream 类及其子类实现,字符输入流由 Reader 类及其子类实现。

(1) InputStream 类

InputStream 类是一个抽象类,是所有基于字节的输入流的超类,InputStream 类的定义如下。

```
public abstract class InputStream implements Closeable;
```

可以看出,InputStream 抽象类继承了 Closeable 类,由于 InputStream 作为一个抽象类,不能用 new 创建实例,所以对字节输入流的操作,都是由它的子类对象完成,它的子类结构如图 5-1 所示。

从图 5-1 中,我们可以看到,字节输入流根据类型和操作方式不同,划分了不同的子类,其中我们主要讲解文件字节输入流(FileInputStream)。

FileInputStream 可以从文件系统中的某个文件中获得输入字节,简单地说,就是用于读取本地文件中的字节数据,FileInputStream 类的构造函数主要有以下三种。

• FileInputStream(File flie):以 file 指定的文件对象创建文件输入流。

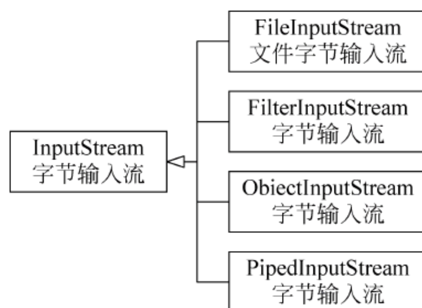


图 5-1 InputStream 类的派生类

- FileInputStream(FileDescriptor fdObj)：以 fdObj 指定的文件描述对象创建文件输入流。
- FileInputStream(String name)：以字符串 name 指定的文件名创建文件输入流。

可以看到，三个构造函数的区别只是参数的不同，所以 FileInputStream 构造方法可以接受字符串、file 对象。其次，它的构造函数在使用的时候需要指定文件的来源，FileInputStream 类的常用方法如表 5-2 所示。

表 5-2 FileInputStream 类的常用方法

方　　法	作　　用
int available()	返回下一次对此输入流调用的方法可以不受阻塞地从此输入流读取(或跳过)的估计剩余字节数
void close()	关闭此文件输入流并释放与此流有关的所有系统资源
protected void finalize()	确保在不再引用文件输入流时调用其 close 方法
int read()	从此输入流中读取一个数据字节
int read(byte[]b)	从此输入流中将最多 b. length 个字节的数据读入一个 byte 数组中
int read(byte[]b, int off,int len)	从此输入流中将最多 len 个字节的数据读入一个 byte 数组中

InputStream 类还提供了许多方法来进行数据读取的操作，常用方法及其作用如表 5-3 所示。

表 5-3 InputStream 类的常用方法

方　　法	作　　用
public int available() throws IOException	获取输入文件的大小
public int close() throws IOException	关闭输入流
public abstract int read() throws IOException	读取内容，一数字的方式读取
public int read(byte[] b) throws IOException	将内容读取到 byte 数组中

我们可以看到，所有的方法都声明抛出异常，所以调用流方法的时候需要进行异常处理。

(2) Reader 类

我们已经介绍了以字节为单位的输入流 InputStream，在这一节中，我们介绍以字符为单位的输入流 Reader 类，以字符方式处理的数据流称为字符流，在 Java 中，存放一个字符需要两个字节，所以字符也是一种特殊的字节流。所有涉及文本的数据处理，例如文本文件、网页和其他常见的文本类型的处理，我们都是使用字符流。

与 InputStream 类一样，Reader 类也是一个抽象类，Reader 类作为字符输入流的顶层类，它的派生类如图 5-2 所示。

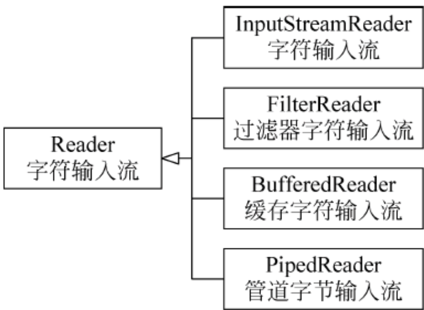


图 5-2　Reader 类派生类

它的常用方法如表 5-4 所示。

表 5-4　Reader 类的常用方法

方　法	作　用
public abstract int close() throws IOException	关闭输出流
public int read() throws IOException	读取单个字符
public int read(char[] cbuf) throws IOException	将内容读取到字符数组中，返回读入的长度

关于 Reader 的子类，较常用的是 BufferReader 子类，它提供通用的缓冲方式文本读取，而且提供了很实用的 readLine，读取一个文本行，从字符输入流中读取文本，缓冲各个字符，从而提供字符、数组和行的高效读取，其构造方法如下。

- BufferReader(Reader in)：创建一个系统默认大小的缓冲字符流。
- BufferReader(Reader in,int size)：创建一个由 size 指定大小的缓冲字符流。

一般使用方法如下：

```
BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream("booklist.txt")));

String data = null;

while((data = br.readLine())!=null)
    {
       System.out.println(data);
    }
```

2. 输出流

与输入流相对应，输出流分为字节输出流和字符输出流。字节输出流由

OutputStream 类及其子类实现，字符输出流由 Write 类及其子类实现。

(1) OutputStream 类

OutputStream 类是字节输出流的父类，它也是一个抽象类，在使用的时候不能直接创建 OutputStream 类对象。它的派生类如图 5-3 所示，常见方法如表 5-5 所示。

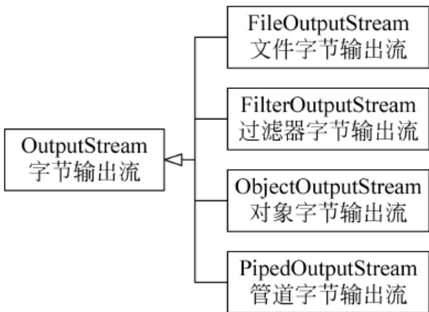


图 5-3　OutputStream 派生类

表 5-5　OutputStream 类的常用方法

方　　法	作　　用
public voidtclose() throws IOException	关闭输出流
public void flush() throws IOException	刷新缓冲区
public void write (byte[] b) throws IOException	将一个 byte 数组写入数据流

在上一节中，我们通过 FileInputStream 对一个文件进行读取，在输出流中，与之对应的是 FileOutputStream 子类。FileOutputStream 用于将字节数据写进文件。它的主要构造方法如下。

- FileOutputStream(File file)：创建一个向指定 File 对象表示的文件中写入数据的文件输出流。
- FileOutputStream(File file，boolean append)：创建一个向指定 File 对象表示的文件中写入数据的文件输出流。
- FileOutputStream(FileDescriptor fdObj)：创建一个向指定文件描述符处写入数据的输出文件流，该文件描述符表示一个到文件系统中的某个实际文件的现有连接。
- FileOutputStream(String name)：创建一个向具有指定名称的文件中写入数据的输出文件流。
- FileOutputStream(String name，boolean append)：创建一个向具有指定 name 的文件中写入数据的输出文件流。

(2) Write 类

Write 类是字符输出流的父类，它也是一个抽象类，它的派生类如图 5-4 所示，常见方法如表 5-6 所示。

表 5-6　Write 类的常用方法

方　　法	作　　用

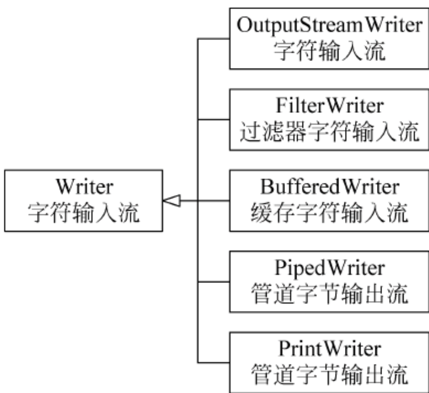


图 5-4 Write 派生类

public abstract void close() throws IOException	关闭输出流
public void write(String str) throws IOException	将字符串输出
public void write (char[] cf) throws IOException	将字符数组输出
public abstract void flush() throws IOException	强制性清空缓存

5.3 文件系统

文件是计算机中一种基本的数据存储形式，它的存储介质有很多，例如硬盘、光盘和U 盘等，Java 中对文件的读写操作通过流实现，它提供了 File 类记载文件属性信息，主要处理与文件或者目录结构的操作。

在上一节中，我们已经基本地介绍了输入输出流的概念，在本节中，我们将通过字节、字符、行等方式实现对文件的写入和写出操作。

5.3.1 按字节读取

Java 中对字节的操作通过 InputStream 和 OutputStream 类，对字节的读取主要通过 InputStream 及其子类实现，并且前面已经介绍了 InputStream 类基本的读取方法。这里，我们主要通过 FileInputStream 类实现以字节读取文件。以字节为单位读取文件，主要用于读取二进制文件，例如图片、声音和影像等文件，其读取步骤如下。

(1) 创建文件对象，读取一个文件，那么需要知道读取的是哪个文件。

```
File file =new File("a.txt");                    //创建 file 对象，读取文件为 a.txt
```

(2) 创建 InputStream 对象，调用 FileInputStream 构造函数。

```
InputStream in=new FileInputStream(file);
```

(3) 设置变量 tempbyte，用于读取结束的标志。

```
int tempbyte;
```

(4) 循环读取文件内容,直到(tempbyte = in.read()) ! = −1。

```
while((tempbyte = in.read())! = -1){
System.out.write(tempbyte);
}
```

(5) 使用 close()关闭输出流,释放资源。

```
in.close();
```

在本例中,只是简单地按照字节打印出文件中的内容,并且是一次读取一个字节,读者可以自行学习如何将读取的内容存入数组,并且如何一次读取多个字节,以及读取过程中如何对异常进行操作,本文不作赘述。

5.3.2 按字符读取

Java 中对字符的操作是通过 Reader 和 Write 及其子类实现的,在前面,我们已经简单介绍了它们的作用和方法,本节中将讲解以字符为单位如何读取文件。

在 Java 中,用字符为单位读取文件常用于读文本、数字等类型的文件,因为字符是由两个字节组成的。在读取过程中,我们可以一次读取一个字节,也可以一次读取多个字节,在本书中,主要通过 Reader 子类 InputStreamReader 类实现对文本的读取,其读取方式如下。

(1) 创建文件对象,读取一个文件,那么需要知道读取的是哪个文件。

```
File file =new File("a.txt");                    //创建 file对象,读取文件为 a.txt
```

(2) 创建 Reader 对象,调用 InputStreamReader 构造函数。

```
InputStream in=new FileInputStream(file);
```

(3) 设置变量 tempchar,用于读取结束的标志。

```
int tempchar;
```

(4) 循环读取文件内容,直到 tempchar = reader.read()) ! = −1。

```
while((tempchar=reader.read())!=-1){
System.out.write(tempbyte);
}
```

(5) 使用 close()关闭输出流,释放资源。

```
reader.close();
```

其中,如上列采用字符读取文件,并且是一次读取一个字节,特别需注意的是,在 Windows 下,r 和 n 两个字符在一起时,表示一次换行,但是当它们分开显示的时候,会换两次行,所以需要采用以下代码,屏蔽 r 或 n,以防出现很多空行:

```
if(((char)tempchar)!='r'){
```

```
System.out.print((char)tempchar);
}
```

5.3.3 按行读取

上面讲解的按字节、字符都是一次读取一个或多个字符,那么在 Java 的文件读取中,我们还可以按行对文件进行读取。对文件进行行读取,主要采用 Reader 类子类 BufferedReader 实现,它提供通用的缓冲方式文本读取,并且提供了 readLine()方法,读取一个文本行,以行为单位读取文件,主要用于面向行的格式化文件。其读取步骤如下。

(1) 创建文件对象,读取一个文件,那么需要知道读取的是哪个文件。

```
File file =new File("a.txt");                    //创建 file对象,读取文件为 a.txt
```

(2) 创建 Reader 对象,调用 BufferedReader 构造函数。

```
reader =  new BufferedReader(new FileReader(file));
```

(3) 设置变量 tempString,用于读取结束的标志,设置 line 变量,用于行号,初始值为 1。

```
String tempString = null;
int line = 1;
```

(4) 循环读取文件内容,一次读入一行,直到读入 null 为文件读取完毕。

```
while ((tempString =  reader.readLine())!=null){
System.out.println("line?"+ line+ ":+ tempString);
line++;
}
```

(5) 使用 close()关闭输出流,释放资源。

```
reader.close();
```

5.3.4 随机读取

在 Java 中,不管是按照字节、字符还是行来存取文件,它们都是顺序性的,都需要一个变量来保存文件当前存取的位置,那么我们是否可以对文件进行随机存取呢? 随机存取是指可以在任何时候将存取文件的指针指向文件内容的任何位置,Java 在 java.io 包中提供了 RandomAccessFile 类,用于处理随机读取的文件。

RandomAccessFile 的唯一父类是 Object,RandomAccessFile 类实现随机存取的关键是它提供的 seek(),它可以任意地指定当前存取文件的指针位置,以便对文件进行随机存取,所以在随机存取文件的时候,我们需要知道文件的大小和位置,指定的单位是字节。

下面简单介绍一下 RandomAccessFile 类的功能和应用。

RandomAccessFile 构造方法如下。

- RandomAccessFile(File file,String mode)：以 file 指定的文件和 mode 指定的读写方式构建对象。
- RandomAccessFile(String name,String mode)：以 name 表示的文件和 mode 指定的读写方式构建对象。

其中,mode 表示文件的读写方式,其含义如下。

- r：读方式,read 简写。用于从文件中读取内容。
- rw：读写方式,对文件可以进行读操作,也可以进行写操作。
- rwd：读写方式,每一次文件内容的修改将被同步写入存储设备。
- rws：读写方式,每一次文件内容的修改和元数据将被同步写入存储设备。

例如,以读写方式打开并写入一行文本：

```
File fis = new File("test.date");
RandomAccessFile raf = new RandomAccessFile(fis, "rw");
byte[] writeStr = "this is a demo!".getBytes();
raf.write(writeStr);
raf.close();
```

RandomAccessFile 类的常用方法如表 5-7 所示。

表 5-7　RandomAccessFile 类的常用方法

方　　法	作　　用
void write(int d)	根据当前指针所在位置处写入一个字节,是将参数 int 的"低 8 位"写出
int read()	如果返回－1 表示读取到了文件末尾 EOF(EOF：End Of File)！每次读取后自动移动文件指针,准备下次读取
void write(byte[] d)	根据当前指针所在位置处连续写出给定数组中的所有字节
void write(byte[] d,int offset,int len)	根据当前指针所在位置处连续写出给定数组中的部分字节,这个部分是从数组的 offset 处开始,连续 len 个字节
int read(byte[]b)	从文件中尝试最多读取给定数组的总长度的字节量,并从给定的字节数组第一个位置开始,将读取到的字节顺序存放至数组中,返回值为实际读取到的字节量
void close()	释放与其关联的所有系统资源
long getFilePointer()	获取当前指针位置
void seek(long pos)	使用该方法可以移动指针到指定位置
int skipBytes(int n)	尝试跳过输入的 n 个字节以丢弃跳过的字节

随机读取文件内容步骤如下。

(1) 打开一个随机访问文件,按只读方式。

```
RandomAccessFilerandomFile=new RandomAccessFile("a.txt","r");
```

(2) 通过 RandomAccessFile 类提供方法 length()获取文件长度。

```
longfileLength= randomFile.length();
```

(3) 读文件的起始位置。

```
intbeginIndex= (fileLength> 4)4:0;
```

(4) 将读文件的起始位置移动到 beginIndex 位置。

```
randomFile.seek(beginIndex);
byte[]bytes= newbyte[10];
intbyteread= 0;
```

(5) 读取文件,一次读取 10 个字节,并且将每次读取的字节数赋值给 byteread。

```
while((byteread= randomFile.read(bytes))!= -1){
System.out.write(bytes, 0, byteread);
}
```

(6) 使用 close()关闭输出流,释放资源。

```
randomFile.close();
```

5.4　图书管理系统 V5.1

在图书管理系统 V5.1 的代码版本中,我们在之前的基础上添加了文件操作,将用户操作的数据读入文件中,对数据进行长期保存,在本项目中,只实现了添加图书和删除图书的操作,其他操作需要读者自行学习,以便了解文件的含义以及用法。

5.4.1　运行效果图

图 5-5 的运行效果图和图 5-6 的 booklist.txt 文件内容图显示的是在 eclipse 中运行程序,添加图书(java,xyp,20),并且通过文件操作将图书信息添加到 booklist.txt 文本中。

5.4.2　类结构示意图

图 5-7 展示的是图书管理系统 V5.1 的类图。

5.4.3　代码实现

Book.java

```
package model;

public class Book {

```

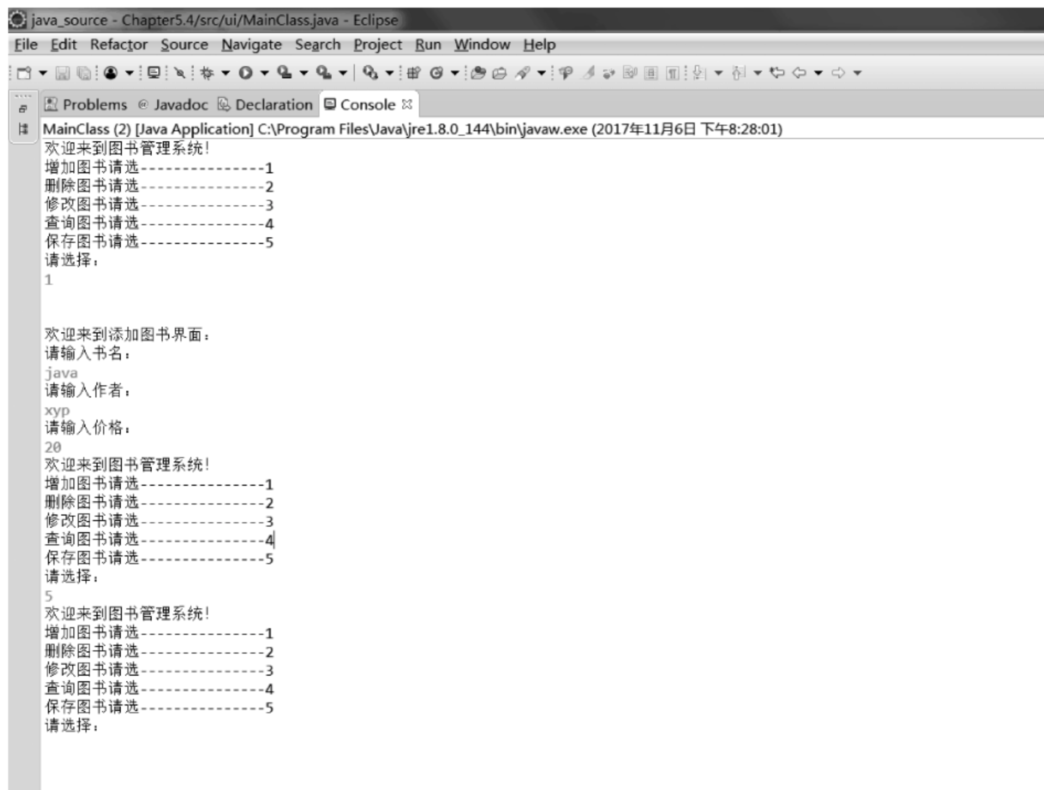


图 5-5　图书管理系统 V5.1 运行效果

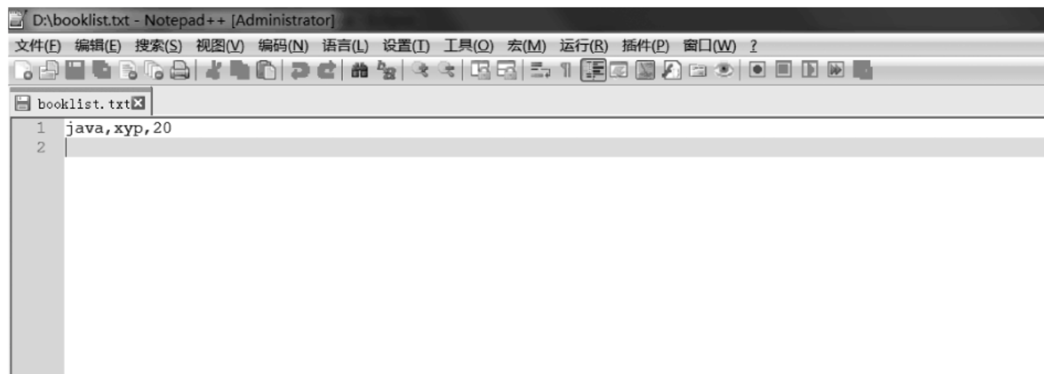


图 5-6　图书管理系统 V5.1 文件 booklist.txt 内容

```
    public String bookname;

    public String author;

    public int price;

    public Book(String bookname, String author, int bookPrice)
    {
```

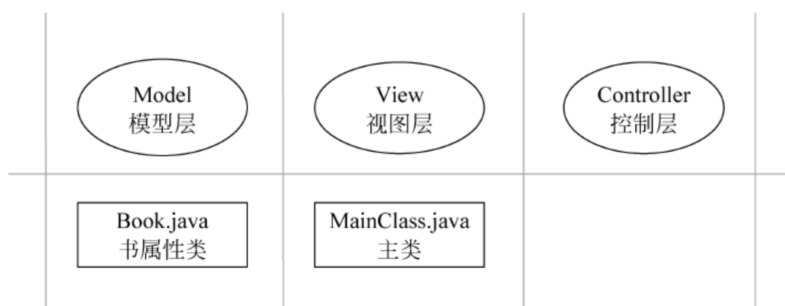


图 5-7　图书管理系统 V5.1 类图

```
        this.bookname = bookname;
        this.author = author;
        this.price = bookPrice;
    }

}
```

MainClass.java

```
package ui;

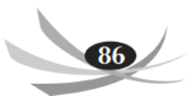
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.util.Scanner;
import java.util.Vector;

import model.Book;

public class MainClass {
    private Vector<Book> bookList = new Vector<>();

    public MainClass() {

        File file = new File("E:/booklist.txt");
        if(file.exists())
        {
            loadData();
        }
```




```
        Scanner input = new Scanner(System.in);
        while (true)
        {
            System.out.println("欢迎来到图书管理系统!");
            System.out.println("增加图书请选---------------1");
            System.out.println("删除图书请选---------------2");
            System.out.println("修改图书请选---------------3");
            System.out.println("查询图书请选---------------4");
            System.out.println("保存图书请选---------------5");
            System.out.println("请选择：");
            int chioce = input.nextInt();
            switch (chioce){
            case 1:
                addBook();
                break;
            case 2:
                deleteBook();
                break;
            case 3:
                changeBook();
                break;
            case 4:
                searchBook();
                break;
            case 5:
                saveData();
                break;
            default:
                break;
            }

        }

    }

    public void addBook()
    {
        Scanner input = new Scanner(System.in);
        System.out.println("\n\n欢迎来到添加图书界面：");

        System.out.println("请输入书名：");
        String bookName = input.nextLine();
        System.out.println("请输入作者：");
        String bookAuthor = input.nextLine();
```

```
        System.out.println("请输入价格：");
        int bookPrice = input.nextInt();

        Book book = new Book(bookName, bookAuthor, bookPrice);
        bookList.add(book);
        System.out.println(bookList.get(0).author);

    }

    public void deleteBook()
    {
        Scanner input = new Scanner(System.in);
        System.out.println("\n\n欢迎来到删除图书界面：");
        System.out.println("请输入需要删除的书名：");
        String bookname = input.nextLine();
        for(int i=0;i<bookList.size();i++)
        {
            if(bookList.get(i).bookname.equals(bookname))
            {
                bookList.remove(i);
            }
        }
    }

    void loadData()
    {
        BufferedReader br= null;
        try
        {
            br= new BufferedReader(new FileReader("E:/booklist.txt"));
            String data=br.readLine();
            while(data!= null)
            {
                String[] str=data.split(",");
                String bookName=str[0];
                String bookAuthor=str[1];
                int bookPrice=Integer.parseInt(str[2]);
                Book book= new Book(bookName,bookAuthor,bookPrice);
                bookList.add(book);
                data=br.readLine();        //接着读下一行
            }
        }
        catch(IOException e)
        {
```

```
            e.printStackTrace();
        }
        finally
        {
            try
            {
                br.close();
            }
            catch(IOException e)
            {
                e.printStackTrace();
            }
        }
    }

    //将操作完的bookList写入TXT文件
    void saveData()
    {
        PrintWriter printWriter=null;
        try
        {
            printWriter=new PrintWriter(new OutputStreamWriter(
            new FileOutputStream("E:/1.txt",false)));
            int bookCount=bookList.size();
            for(int i=0;i<bookCount;i++)
            {
                printWriter.println(bookList.get(i).bookname+","
                +bookList.get(i).author+","
                +bookList.get(i).price);
            }
            printWriter.flush();
        }
        catch(FileNotFoundException e)
        {
            e.printStackTrace();
        }
        finally
        {
            printWriter.close();
        }
    }

    public void changeBook()
    {
```

```
    }
    public void searchBook()
    {

    }
    public static void main(String[] args) {
        new MainClass();
    }
}
```

5.5 数 据 库

在前面的章节中,我们已经介绍了 Java 基本数据存储的方法,但是在实际的项目中,往往会涉及大量的数据存储以及访问的问题,我们就需要使用数据库来存储项目中使用到的大量数据,以方便对其进行访问和修改。在本节中,我们将介绍 JDBC 以及 JDBC 在数据库连接中的使用方法和作用。

5.5.1 JDBC简介

JDBC 的全称是 Java DataBase Connectivity,也就是 Java 数据库连接,简单来说,JDBC 是一种用于执行 SQL 语句的 Java API,是 Java 与数据库之间连接的桥梁。Java 程序通过调用 JDBC 提供的接口和类所提供的方法,使得用户能够以相同的方式连接不同的数据库系统。

JDBC 由一组 Java 语言编写的类和接口组成,并且所有的类和接口都放在 java.sql 中,JDBC 主要用来执行 SQL 查询,存储过程,并处理返回的结果。

在使用 JDBC 之前,我们可以查看 JDBC 的基本结构,如图 5-8 所示。在 JDBC 的基本结构中,顶层是 Java 应用程序(Java Application),就是我们需要连接数据库的一切 Java 程序。JDBC 由 JDBC API 和 JDBC Driver Interface 组成,JDBC API 是应用程序连接数据库的接口,JDBC Driver Interface 是面向 JDBC 驱动程序开发商的编程接口,它会把我们通过 JDBC API 发给数据库的通用指令翻译给它们自己的数据库。其下的 JDBC 驱动程序是在 JDBC API 中实现的接口,用于与数据库服务器进行交互。其次,JDBC 提供了四种类型的驱动程序,分别是 JDBC-JOBC 桥、本地 API、网络协议驱动和本地协议驱动,在这里就不作一一讲述了,读者可以自行学习。

5.5.2 JDBC访问数据库的基本过程

在前面,我们了解了 JDBC 的层次结构和工作原理,JDBC 为我们在程序中操作数据库提供了良好的前提,在本节中,我们将介绍 JDBC 访问数据库的基本过程。在本书中,我们使用的是 MySQL 数据库,所以本节中,我们只讲解 JDBC 连接 MySQL 数据库的过程,读者可以自行查看 JBDC 连接 Assess、Oracle 等其他数据库的连接方法和步骤。

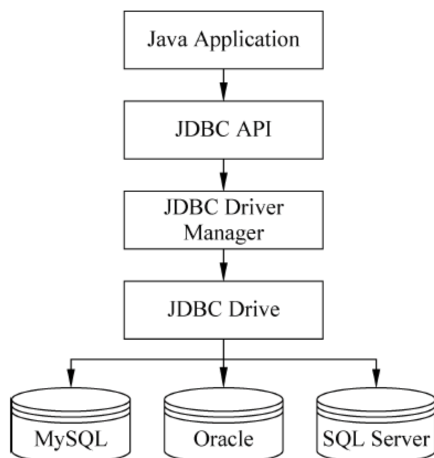


图 5-8 JDBC 体系图

利用 JDBC 访问 MySQL 数据库一般需要以下几个基本步骤。

1. 加载 JDBC 驱动程序。
2. 创建数据库连接。
3. 创建 Statement 操作对象。
4. 向数据库发送执行 SQL 的语句。
5. 处理查询结果集。
6. 关闭 JDBC 对象。

具体实现步骤如下。

(1) 加载 JDBC 驱动程序

Java 连接 Mysql 数据库需要驱动包，驱动包可以从 MySQL 官网下载，解压后得到相应的 jar 包，将 jar 包导入项目中即可，如图 5-9 所示。

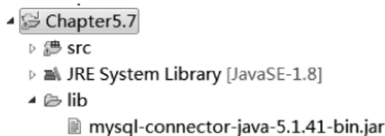


图 5-9 MySQL 架包导入

然后右击工程名，在 java build path 中的 Libraries 分页中选择 Add JARs...，选择刚才添加的 JDBC，如图 5-10 所示。

将驱动包加载好之后，在连接数据库的类中，还需要添加数据库驱动程序，连接不同的数据库，加载的驱动程序不同，MySQL JDBC 驱动程序的类名是 org. git. mm. mysql. Driver。在本书项目中，连接 MySQL 数据库都是通过 Class. forName(com. mysql. jdbc. Driver)加载驱动，这段代码返回一个 Driver 对象，在返回的过程中通过执行以下代码实现驱动的加载：

```
java.sql.DriverManager.registerDriver(newDriver());
```

(2) 创建数据库连接

加载完驱动程序之后，就可以创建应用程序和数据库之间的连接了，我们常常通过 DriverManager 类提供静态方法 getConnection()来创建程序与数据库之间的连接，与 MySQL 数据库建立连接的代码如下：

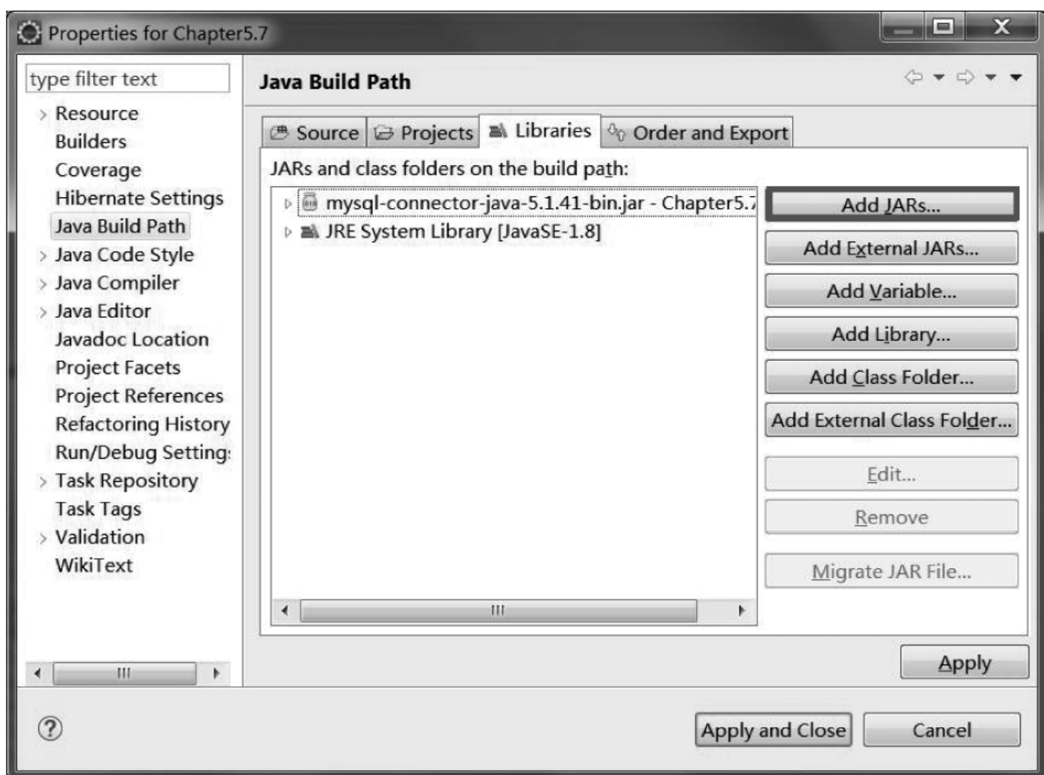


图 5-10　eclipse 导入 MySQL 架包

```
Connection conn=DriverManager.getConnection(url, user, password);
```

在上面代码中，url、user、password 是连接数据路必须的参数，其中 user、password 是我们安装 MySQL 数据库时设置的账号和密码。url 指向的是相应的数据库，MySQL 基本格式如下：

jdbc：mysql：//hostname：port/具体数据库名，其中 hostname 和 port 对应数据库用户名密码。

（3）创建 Statement 操作对象

通过以上步骤，应用程序和数据库连接成功，就需要创建 SQL 语句对象，用来执行用户定义的 SQL 语句，这里，我们需要创建 Statement 操作对象，这需要使用 Connection 对象提供的 createStatement()方法实现，其定义如下：

```
Public Statement createStatement() throws SQLExcetion;
```

该方法返回建立的 Statement 对象，我们在使用中，都需要对其进行异常处理，当出现问题的时候，它将会抛出 SQLExcetion 异常。这其实是创建 Statement 的一种方法，Connection 提供了三种方法，也可以通过 Statement 的子类接口创建 SQL 语句对象。

（4）向数据库发送执行 SQL 的语句

创建了 Statement 操作对象之后，接下来就是用户定义的 SQL 语句向数据库发送并

执行。由于SQL语句分为查询语句、数据定义和更新语句，所以Statement对象也提供了不同的执行SQL语句的方法，以下是两种常用的方法。

- ResultSet executeQuery(String sql)；
- int executeUpdate(String sql)；

其中，executeQuery()方法用于执行SQLECT查询语句，executeUpdate()方法用于执行改变数据库内容的SQL，如update、Insert、delete等数据定义和更新语句。

例如，下面代码是删除booklist表中书编号为001的代码片段：

```
String sql="detele from booklist where bookId="001"";
int n=sta.executeUpdate(sql);
```

(5) 处理结果集

根据SQL语句的不同类型，结果集分为以下两种情况。

- 执行更新返回的是本次操作影响到的记录数。
- 执行查询返回的结果是一个ResultSet对象。

其中，针对查询结果集，它返回的是一个ResultSet结果集对象，每一个ResultSet对象都有一个游标指向结果集的当前对象，最初光标被置于第一行之前，使用ResultSet对象提供的方法next()可以改变游标的位置，以获取每一行数据。当next()返回false时，说明记录已经全部遍历，程序也可以退出。那么针对每一行的数据，它由多个数据构成，例如查询书本信息，一行就包含书名、作者、价格等信息，我们又通过什么方法获取每一个数组以存储到booklist中呢？ResultSet对象提供了getXXX(int columnIndex)和getXXX(String columnLabel)两种方法获取结果集中每一行中的每一个记录。其中，getXXX(int columnIndex)方法使用列索引获取值，列从1开始。getXXX(String columnLabel)方法使用列的名称获取值。如下代码分别使用getXXX(int columnIndex)、getXXX(String columnLabel)获取查询结果中的每一本书籍内容。

- 使用getXXX(int columnIndex)实现

```
while (rs.next())
{
String bookname = rs.getString(1);
String author1 = rs.getString(2);
int price = rs.getInt(3);
}
```

- 使用getXXX(String columnLabel)实现

```
while (rs.next())
{
String bookname = rs.getString("bookname");
String author1 = rs.getString("author");
int price = rs.getInt("bookPrice");
}
```

(6) 关闭 JDBC 对象

对数据库操作完成之后，通常需要调用 ResultSet 对象、Statement 对象、Connection 对象提供的 close()方法将各自的对象关闭，释放 JDBC 资源，并且关闭顺序与声明顺序相反，关闭顺序依次是记录集、操作对象、连接对象。

5.5.3　JDBC 常用类

我们认识了 JDBC 连接 MySQL 数据库的基本步骤，那么在数据库连接成功之后，我们就需要在程序中对数据库进行操作，所以，我们需要对 JDBC 的常用类和接口进行了解和学习。

在本书中，我们主要介绍 DriverManager 类、Connection 接口、Statement 接口、ResultSet 接口、PreparedStatent 接口。

1. DriverManager 类

DriverManager 类是 JDBC 的管理层，是管理一组 JDBC 驱动程序的基本服务，用来管理数据库中所有的驱动程序，作用于用户和驱动程序之间，跟踪可用的驱动程序，并在数据库的驱动程序之间建立连接。DriverManager 类中定义的主要方法如表 5-8 所示。

表 5-8　DriverManager 类的主要方法

方　法	作　用
static Connection getConnection(String url)	创建与指定的数据库 url 的连接
staticConnection getConnection(String url, String user, String password)	试图建立到给定数据库 url 的连接 url 格式：jdbc:subprotocol:subnameuser：数据库用户名 password：用户的密码
public staticConnection getConnection(String url, Propertiesinfo)	通过指定的数据库 url 及其属性信息创建数据库 info：一系列字符串键值对用来作为连接参数，一般至少包括 user 和 password 两个属性
staticDriver getDriver(String url)	查找能打开 url 所指定的数据库的驱动程序
static void deregisterDriver(Driver driver)	从 DriverManager 的列表中删除一个驱动程序。applet 只能注销取自其自身的类加载器的驱动程序 drive：要删除的 JDBC Driver

表 5-8 中只是介绍了几个常用的 DriverManager 类的方法，在实际程序开发中，我们可以根据自己的需要查看 API 中方法及其含义。

通过表 5-8 我们可以看到，DriverManager 类中的方法都是静态方法，所以在程序中不用再对它进行实例化，直接通过类名调用。在表 5-8 中，我们看到 DriverManager 类提供了三种建立数据库连接的方法，这三种方法都返回一个 Connection 对象实例，只是传递参数不同，一般，我们常用的连接方式是 getConnection(String url, String user, String password)。这三种方法中的参数 url 用于指定数据源和用于连接到该数据源的数据库的连接类型，并且在数据库连接中，对于不同的数据库其内容不相同。

2. Connection 接口

Connection 接口是 java.sql 包中定义的接口，它的主要作用是建立与数据库的连接，

并且在连接上下文中执行 SQL 语句并返回结果，所以 Connection 最为重要的一个方法就是用来获取 Statement 对象。Connection 提供的常用方法如表 5-9 所示。

表 5-9　Connection 类的主要方法

方　　法	作　　用
Statement createStatement()	创建一个 Statement 对象来将 SQL 语句发送到数据库
Statement createStatement(int resultSetType, int resultSetConcurrency)	创建一个 Statement 对象，该对象将生成具有给定类型和并发性的 ResultSet 对象
void close()	立即释放此 Connection 对象的数据库和 JDBC 资源
boolean isClosed()	判断连接是否已关闭
String getCatalog()	获取连接对象的当前目录名

3. Statement 接口

当我们使用 Connection 接口建立了程序与数据库之间的连接之后，我们就可以对数据库进行操作了，在对数据库的操作中，最常见的就是 SQL 语句的使用，那么 Statement 就是在已建立数据库连接的基础上，向数据库发送要执行的 SQL 语句，简单地说，Statement 就是执行静态 SQL 语句并返回它所生成结果的对象。

在实际使用中，实际上有三种 Statement 对象。首先就是 Statement 对象，Statement 对象用于执行不带参数的简单 SQL 语句，并且 Statement 接口提供了执行语句和获取结果的基本方法；其次是 PreparedStatement 对象，它是继承 Statement 而来的，主要执行带或不带 IN 参数的预编译 SQL 语句，PreparedStatement 接口添加了处理 IN 参数的方法；最后是 CallableStatement，CallableStatement 对象是继承 PreparedStatement 而来的，主要用于执行对数据库已存储过程的调用，相应地，CallableStatement 接口添加了处理 OUT 参数的方法，在实际情况中，如何使用 Statement 对象，还需要根据不同的需求进行选择。

正如上面所讲，Statement 接口定义了执行 SQL 语句和获取返回结果的成员方法，Statement 接口中定义的主要方法如表 5-10 所示。

表 5-10　Statement 接口的主要方法

方　　法	作　　用
void close()	立即释放此 Statement 对象的数据库和 JDBC 资源
boolean execute(String sql)	执行给定的 SQL 语句，该语句可能返回多个结果
ResultSet getResultSet()	以 ResultSet 对象的形式获取当前结果。每个结果只应调用一次此方法
ResultSet executeQuery(String sql)	执行给定的 SQL 语句，该语句返回单个 ResultSet 对象
int executeUpdate(String sql)	进行数据库更新的 SQL 语句，包括 INSERT、UPDATE、DETELE 或 SQL DDL 等语句

4. ResultSet 接口

结果集 ResultSet 是用来暂时存放执行 SQL 语句后产生的结果的集合，简单来说，就是处理数据库操作的结果，它不仅包含了符合 SQL 语句中条件的所有行，并且也提供了一套 get 方法对行中数据进行访问。ResultSet 接口的主要方法如表 5-11 所示。

表 5-11　ResultSet 接口的主要方法

方　法	作　用
boolean absolute(int row)	将指针移动到结果集对象的某一行
void afterLast()	将指针移动到结果集对象的末尾
void beforeFirst()	将指针移动到结果集对象的头部
boolean first()	将指针移动到结果集对象的第一行
boolean next()	将指针从当前位置移动到下一行
int getInt(int columnIndex)	获取当前行中某一列的值,返回一个整型值
void insertRow()	新增记录到数据库中
void deleteRow()	从此 ResultSet 对象和底层数据库中删除当前行

在表 5-11 中,我们只是列举了 ResultSet 接口几个常用的方法,读者可以通过 API 查看其他的方法及其含义,ResultSet 类方法的作用可以简单地划分为改变指针位置、获取列的值、更新结果集。

5.6　MVC 设计模式

5.6.1　什么是 MVC 设计模式

MVC 模式(三层架构模式)(Model-View-Controller)是软件工程中的一种软件架构模式,将软件系统分为三个基本部分:控制器(Controller)、视图(View)和模型(Model)。

控制器(Controller):负责转发请求,对请求进行处理。

视图(View):界面设计人员进行图形界面设计。

模型(Model):程序员编写程序应有的功能(实现算法等),数据库专家进行数据管理和数据库设计(可以实现具体的功能)。

5.6.2　为什么要使用 MVC 设计模式

MVC 实现了视图层和业务层分离,这样就允许更改视图层代码而不用重新编写模型和控制器代码,同样,一个应用的业务流程或者业务规则的改变只需要改动 MVC 的模型层即可。因为模型与控制器和视图相分离。这种分离有许多好处:

(1) 清晰地将应用程序分隔为独立的部分;

(2) 业务逻辑代码能够很方便地在多处重复使用;

(3) 方便开发人员分工协作;

(4) 如果需要,可以方便开发人员对应用程序各个部分的代码进行测试。

5.7　图书管理系统 V5.2

在图书管理系统 V5.2 的章节代码中,我们主要将设计结构修改为 MVC 的模式,MVC 模式使我们更加方便地掌握代码结构,并且提高了可阅读性。

5.7.1 运行效果图

图书管理系统 V5.2 运行结果与前面相同，只是在操作过程中，数据存储在数据库中，此处不作显示。

5.7.2 类结构示意图

图 5-11 展示了图书管理系统 V5.2 的类图。

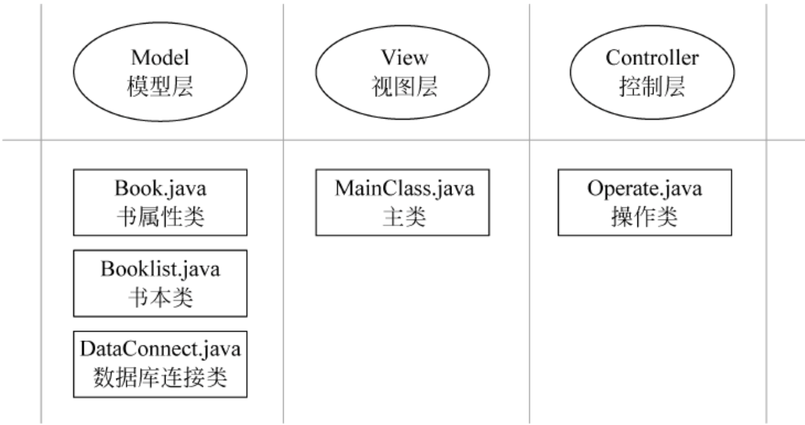


图 5-11　图书管理系统 V5.2 类图

5.7.3 代码实现

Book.java

```
package model;

public class Book {

    public String bookname;

    public String author;

    public int price;

    public Book(String bookname, String author, int bookPrice)
    {
        this.bookname = bookname;
        this.author = author;
        this.price = bookPrice;
    }
}
```

Booklist.java

```
package model;
```

```

import java.util.Vector;

public class Booklist extends Vector<Book> {

}
```

DataConnect.java

```
package dbConnect;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DataConnect {
        static final String url = utf8&useSSL=true";
        "jdbc:mysql://localhost:3306/book?characterEncoding=
        static final String name = "com.mysql.jdbc.Driver";
        static final String user = "root";
        static final String password = "mysql123";
        static
        {
            try
                {
                    Class.forName(name);  //指定连接类型
                    System.out.println("lianije");
                }
            catch(ClassNotFoundException e)
                {
                    System.out.println("Class Not Found Exception:"+e.toString());
                }
        }

        public static Connection getConnection()
        {
            try {
                return DriverManager.getConnection(url, user, password);
                } catch (SQLException e) {
                e.printStackTrace();
                return null;
            }
        }

```

```
        public static void closeConnection(ResultSet rs,Statement statement,Connection con)

        {
                try {
                    if(rs!=null)rs.close();
                    if(statement!=null)statement.close();
                    if(con!=null)con.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
        }

        public static void closeConnection(Statement statement,Connection con)
        {
            closeConnection(null,statement,con);
        }

}


```

MainClass.java

```
package ui;

import java.io.IOException;
import java.util.Scanner;
import java.util.Vector;

import control.Operate;
import model.Book;

public class MainClass {

    private Vector<Book> bookList = new Vector<>();

    public MainClass() {
    Scanner input = new Scanner(System.in);
    Operate operate = new Operate();
        while (true)
        {
            System.out.println("欢迎来到图书管理系统!");
            System.out.println("增加图书请选----------------1");
            System.out.println("删除图书请选----------------2");
            System.out.println("修改图书请选----------------3");
            System.out.println("查询图书请选----------------4");
```

```
                System.out.println("请选择：");
                int chiooe = input.nextInt();
                switch (chiooe){
                case 1:
                    System.out.println("\n\n欢迎来到添加图书界面：");
                    System.out.println("请输入书名：");
                    String bookName = input.nextLine();
                    System.out.println("请输入作者：");
                    String bookAuthor = input.nextLine();
                    System.out.println("请输入价格：");
                    int bookPrice = input.nextInt();
                    Book book = new Book(bookName, bookAuthor, bookPrice);
                    operate.addBook(book);
                    break;
                case 2:
                    Scanner inputname = new Scanner(System.in);
                    System.out.println("\n\n欢迎来到删除图书界面：");
                    System.out.println("请输入需要删除图书的名字：");
                    String bookname = inputname.nextLine();
                    operate.deletebook(bookname);
                    break;
                case 3:
                    break;
                case 4:
                    break;
                default:
                    break;
                }
            }
        }
        public static void main(String[] args) {
            new MainClass();
        }
    }
```

Operator.java




```
package control;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import dbConnect.DataConnect;
import model.Book;
import model.Booklist;

public class Operator {
        public boolean addBook(Book book)
        {
            Connection conn = DataConnect.getConnection();
            Statement s;

            try {
                s = conn.createStatement();
                String sql = "insert into booklist (bookname,author,price)
                values('"+book.bookname+"','"+book.author+"',"+book.price+")";
                boolean isSuccess = s.execute(sql);
                return isSuccess;
          } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return false;
        }

        public boolean deletebook(String bookname)
        {
            Connection conn = DataConnect.getConnection();
            try {

                Statement s = conn.createStatement();
                String sql = "delete from booklist where bookname =
                '"+bookname+"'";
                boolean isSuccess = s.execute(sql);
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return false;
```

```
    }

    public boolean updatename(String bookname1,String bookname2)
    {
        Connection conn = DataConnect.getConnection();
        try {
            Statement s = conn.createStatement();
            String sql = "update booklist set bookname='"+bookname2+"'where
            bookname='"+bookname1+"' ";
            boolean isSuccess = s.execute(sql);
      } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return false;
    }

    public boolean updateauthor(String author1, String author2)
    {
        Connection conn = DataConnect.getConnection();
        try {
            Statement s = conn.createStatement();
            String sql = "update booklist set author='"+author2+"'where
            author='"+author1+"' ";
            boolean isSuccess = s.execute(sql);
      } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return false;
    }

    public boolean updateprice(float price1, float price2)
    {
        Connection conn = DataConnect.getConnection();
        try {
            Statement s = conn.createStatement();
            String sql = "update booklist set price='"+price2+"'where
            price='"+price1+"' ";
            boolean isSuccess = s.execute(sql);
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

```
            return false;
        }

        public Booklist searchAll()
        {
            Booklist booklist = new Booklist();
            Connection conn = DataConnect.getConnection();
            Statement s;
            try {
                s = conn.createStatement();
                String sql = "select * from booklist ";
                ResultSet rs = s.executeQuery(sql);
                while (rs.next())
                {
                    String bookname = rs.getString(1);
                    String author = rs.getString(2);
                    int price = rs.getInt(3);
                    Book book = new Book(bookname,author,price);
                    booklist.add(book);
                }
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return booklist;
        }

        public Booklist searchname(String bookname)
        {
                    Booklist booklist = new Booklist();
                    Connection conn = DataConnect.getConnection();
                    Statement s;
                    try {
                        s = conn.createStatement();
                        String sql = "select * from booklist where bookname =
                        '"+bookname+"'";
                        ResultSet rs = s.executeQuery(sql);
                        while (rs.next())
                        {
                             String bookname1 = rs.getString(1);
                            String author = rs.getString(2);
                            int price = rs.getInt(3);
                            Book book = new Book(bookname1,author,price);
                            booklist.add(book);
```

```
                }
                return booklist;
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return null;
    }

    public Booklist searchauthor(String author)
    {
        Booklist booklist = new Booklist();
        Connection conn = DataConnect.getConnection();
        Statement s;
        try {
            s = conn.createStatement();
            String sql = "select * from booklist where author = '"+author+"'";
            ResultSet rs = s.executeQuery(sql);
            while (rs.next())
            {
                String bookname = rs.getString(1);
                String author1 = rs.getString(2);
                int price = rs.getInt(3);
                Book book = new Book(bookname,author1,price);
                booklist.add(book);

            }
            return booklist;
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return null;
    }

    public Booklist searchprice(float price)
    {
        Booklist booklist = new Booklist();
        Connection conn = DataConnect.getConnection();
        Statement s;
        try {
            s = conn.createStatement();
            String sql = "select * from booklist where price = 'price'";
            ResultSet rs = s.executeQuery(sql);
```

```
                while (rs.next())
                {
                    String bookname = rs.getString(1);
                    String author = rs.getString(2);
                    int price1 = rs.getInt(3);
                    Book book = new Book(bookname,author,price1);
                    booklist.add(book);
                }
                return booklist;
            } catch (SQLException e) {
                 // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return null;
        }
        public static void main(String[] args) {
            // TODO Auto-generated method stub

        }

    }

```

细心的读者也许已经发现，在 Operator 这个类里直到使用完数据库连接，都没有调用 DataConnect. closeConnection(…)将连接关掉，没有做好善后工作。这是不对的，用完连接应该把连接关掉、释放掉才对。不仅是这一章，后面章节的代码都请读者加上，不要忘了用 try…catch…finally，把关闭连接的代码放在 finally 里面。

第 6 章

界　面

进入本章，我们将会接触到可视化的窗口界面，通过 Java 提供的 API 可对 Java 可视化窗口有一定的了解。

6.1　本章任务

理论任务：了解界面的基本原理，掌握界面编程技术。

实践任务：把图书管理系统白底黑字的界面换成漂亮的界面。

6.2　画画的故事

相信很多读者都有过画画的经历，在画画的时候我们首先需要准备一张画纸，这张画纸其实就是 Java GUI(Graphics User Interface)里面的容器。当然，要画出一幅完美的画，还需要对纸张进行规划，左边画什么，右边画什么，中间又画什么？这种规划，在 Java GUI 里面我们称之为布局管理器。最后，最重要的部分，就是要开始作画了，画在画纸上的各个内容，在 Java GUI 中叫作组件，如图 6-1 所示。其实 Java GUI 应用程序设计的思想与画画是类似的，接下来将详细介绍与 GUI 应用程序设计有关的知识。

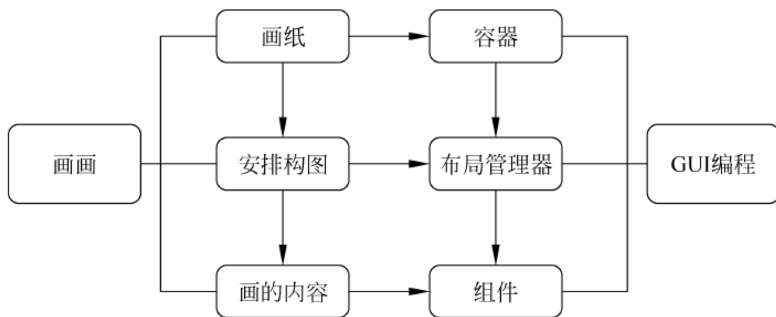


图 6-1　画画与 GUI 编程

6.3　容　　器

在讲容器之前，先讲一下 GUI 编程经常提到的知识点。Java 的 java.awt 包，即 Java 抽象窗口工具包(Abstract Window Toolkit，AWT)，它提供了许多用来设计 GUI 的组件

类。Java 早期进行用户界面设计时，主要使用 java. awt 包提供的类，JDK1. 2 推出之后，增加了一个新的 javax. swing 包，该包提供了功能更加强大的用来设计 GUI 的类。简单地讲，大家可以认为 Swing 类是 GUI 界面类的升级版。java. awt 和 javax. swing 包中一部分类的层次关系的 UML 类图如图 6-2 所示。

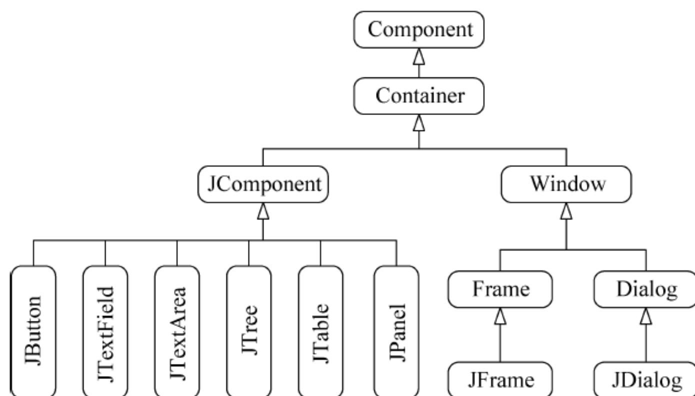


图 6-2 Component 类的部分子类

Java 把 Component 类的子类或间接子类创建的对象称为一个组件；把 Container 的子类或间接子类创建的对象称为一个容器。可以向容器添加组件，容器本身也是一个组件，因此可以把一个容器添加到另一个容器中实现容器的嵌套。

容器是 GUI 设计中必不可少的一种界面元素，它是用来放置其他组件的一种特殊部件，Java 类库中提供了丰富的容器类，为选择与创建容器带来了极大的便捷。下面介绍两种常用容器：底层容器和面板容器。

6.3.1 底层容器

底层容器是指最外层的容器，即包含所有组件或容器的那层容器。例如，运行应用程序后打开的最外层窗口。Java 提供的 JFrame 类的实例，即通常所说的窗口就是一个底层容器；JDialog 类的实例，即通常所说的对话框也是一个底层容器。每一个可视化的 GUI 应用程序都应该有一个底层容器，其他组件必须被添加到底层容器中，以便借助这个底层容器和操作系统进行信息交互。

下面分别介绍底层容器的创建、定位和调整大小的基本方法。

1. 创建底层容器

通常，底层容器就是人们看到的最外层窗口，创建这个窗口的基本过程如下。

(1) 定义一个 JFrame 的子类。

(2) 创建上述子类的对象。

(3) 设置窗口关闭操作。

【例 6-1】 创建底层容器。

下面是定义 JFrame 子类的程序代码。

```
import javax.swing.*;
```

```

public class SimpleJFrameClass extends JFrame{
    public static final int DEFAULT_WIDTH=300;
    //窗口默认宽度
    public static final int DEFAULT_HEIGHT=200;
    //窗口默认高度
    public SimpleJFrameClass (){
        setSize(DEFAULT_WIDTH,DEFAULT_HEIGHT);
        //设置窗口大小
        setTitle("Simple JFrame Window");
        //设置标题栏
        setVisible(true);
        //设置可见性
    }
}
```

下面是检测 SimpleJFrameClass 类使用情况的测试类 TestSimpleJFrameClass 的程序代码。

```
import javax.swing.*;

    public class TestSimpleJFrameClass{
    public static void main(String[] agrs){
        SimpleJFrameClass frame=new SimpleJFrameClass();
        //创建窗口
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //设置关闭窗口操作
    }
}

```

2. 定位底层容器

JFrame 类从各层父类中继承了许多用于处理窗口大小及位置的方法。其中,定位底层容器的成员方法主要有 setLocation()与 setBounds()。

(1) setLocation 的定义格式为:

```
public void setLocation(int x,int y);
```

该成员方法的功能是将底层窗口移至屏幕坐标(x,y)的位置。

(2) setBounds 的定义格式为:

```
public void setBounds(int xleft,int yleft,int width,int height);
```

该成员方法的功能是将底层窗口的左上角移至屏幕坐标为(xleft,yleft)处,宽为 width,高为 height。

3. 设置底层容器大小

前面已经介绍过 setSize()用于设置底层容器的大小,另外,还可以利用 setResiable()通

过带入 true 或 false 确定底层容器的大小是否可调节。

6.3.2 面板容器

面板是一种没有边框、没有标题栏的中间层容器。常见的面板容器有两种：一种是普通的面板容器，在 Swing 中用 JPanel 类实现；另一种是带滚动视图的容器，在 Swing 中用 JScrollPane 类实现。下面分别介绍这两种面板容器的使用方式。

1. 普通面板的容器

这是一种常用的容器种类。经常使用 JPanel 先创建一个面板，再向这个面板添加组件，然后把这个面板添加到其他容器中。JPanel 面板的默认布局是 FlowLayout 布局（布局管理器将在 6.4 节中讲解）。

（1）在 JPanel 类中提供了两种格式的构造方法

- JPanel()：无参构造方法，它将创建一个布局管理器为 FlowLayout 的面板容器。
- JPanel(LayoutManager layout)：这个构造方法将创造一个布局管理器为 layout 的面板容器。

（2）使用 JPanel 面板容器的基本过程

- 定义一个 JPanel 的子类。
- 创建上述子类的对象 panel。
- 使用 getContentPane().add(panel)方法将面板放置到窗口中。

实际上，外层窗口的内容窗格也是一种没有边框的面板容器，利用成员方法 getContentPane()可以获取它。

2. 带滚动视图的容器

由于屏幕大小的限制，有些组件不能在一屏中全部显示出来或显示内容的大小不能动态地发生变化，此时，可以使用带滚动功能的视图容器。滚动窗格只可以添加一个组件，可以把一个组件放到一个滚动窗格中，然后通过滚动条来观看该组件。JTextArea 不自带滚动条，因此需要把文本区放到一个滚动窗格中。例如：

```
JScrollPane scroll = new JScrollPane(new JTextArea());
```

除了以上介绍的常用的容器以外，还有很多其他的容器，建议读者在学习时，要善于查阅 Java 提供的类库帮助文档，例如下载 Java 类库帮助文档 jdk-7-doc.zip。

6.4 布局管理器

在 6.3 节中，讲述了容器的基本概念和将组件放入容器的基本方法。但是，如何在容器中摆放各个组件，每个组件的大小如何控制是界面设计者需要熟练掌握的内容。本节将介绍 Java 布局组件的基本策略和所涉及的相关类。

6.4.1 布局管理器概述

布局管理器是指按照特定的策略安排每个组件在容器中摆放位置及大小的一种特殊对象。容器可以使用 setLayout(布局对象)方法设置自己的布局。Java 提供的布局管理

器的类层次结构如图 6-3 所示。

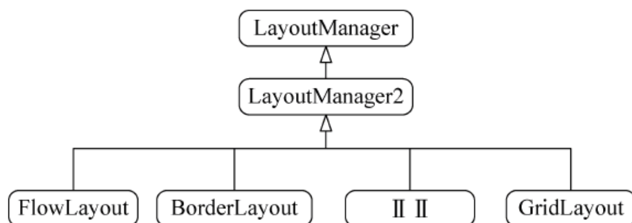


图 6-3 Java 提供的布局管理器的类层次结构

6.4.2 FlowLayout 布局管理器

FlowLayout 被称为流程布局管理器，它是 JPanel 面板容器的默认布局管理器，即 JPanel 及其子类创建的容器对象，如果不专门为其指定布局，则它们的布局就是 FlowLayout 型布局。

在 FlowLayout 类中，提供了以下 3 种格式的构造方法。

(1) FlowLayout()：无参构造方法。它将创建一个对齐方式为居中，水平和垂直间距为 5 个像素的布局管理器对象。

(2) FlowLayout(int align)：这个构造方法将创建一个对齐方式为 align 的布局管理器对象。

(3) FlowLayout(int align, int hgap, int vgap)：这个构造方法将创建一个对齐方式为 align，水平间隙为 hgap 个像素和垂直间隙为 vgap 个像素的布局管理器对象。

6.4.3 BorderLayout 布局管理器

BorderLayout 被称为边框布局管理器，它是 JFrame 内容窗格的默认布局管理器。这个布局管理器把容器的布局分为五个位置：CENTER、EAST、WEST、NORTH、SOUTH，如图 6-4 所示。

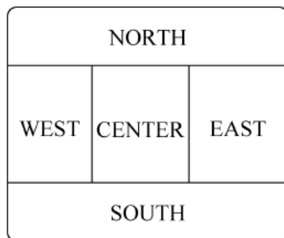


图 6-4 BorderLayout 布局管理器的布局方式

在 BorderLayout 类中，提供了以下两种格式的构造方法。

(1) BorderLayout()：构造一个组件之间没有间距(默认间距为 0 像素)的新边框布局。

(2) BorderLayout(int hgap, int vgap)：构造一个具有指定组件(hgap 为横向间距，vgap 为纵向间距)间距的边框布局。

6.4.4 GridLayout 布局管理器

GridLayout 被称为网格布局管理器，它是一种非常容易理解的布局管理器。这种布局管理器的特点是将容器按照指定的行数、列数分成大小相等的网格。

在 GridLayout 类中，提供了以下 3 种格式的构造方法。

(1) GridLayout()：创建具有默认值的网格布局，即每个组件占据一行一列。

(2) GridLayout(int rows, int cols)：创建具有指定行数和列数的网格布局，Rows 为行数，cols 为列数。

(3) GridLayout(int rows, int cols, int hgap, int vgap)：创建具有指定行数、列数以及组件水平、纵向一定间距的网格布局。

除上面介绍的 3 种布局管理器以外，Java 语言的类库还提供了几种功能更加强大的布局管理器，尽管不同的布局管理器的布局策略有所不同，但基本的使用方式是相同的，读者可以查阅 Java 文档，了解有关这些布局管理器的相关内容。

6.5 组　　件

组件是应用程序界面中的重要组成元素，丰富的组件种类构成了强大的软件开发资源。在程序开发过程中，根据不同的需求，选择适合的组件是一件技术性很强的工作，它关系到应用程序界面的美观性、适用性、方便性和安全性。

6.5.1 组件概述

在 Swing 中，所有组件都是 JComponent 类的子类。JComponent 不仅从 Component 和 Container 类继承了大量的成员方法，还增加了部分成员方法，它们分别用来实现定制组件外观、设置或获取组件状态、处理事件、绘制组件、处理组件层次、布局组件、获得组件大小和放置位置、指定组件绝对大小和位置等。

6.5.2 常用的组件

1. 文本框

使用 JTextField 类创建文本框，允许用户在文本框中输入单行文本。

2. 文本区

使用 JTextArea 类创建文本区，允许用户在文本区中输入多行文本。

3. 按钮

使用 JButton 类创建按钮，允许用户单击按钮。

4. 标签

使用 JLabel 类创建标签，标签为用户提供信息的提示。

5. 复选框

使用 JCheckBox 类创建复选框，为用户提供多项选择。复选框的右边有个名字，并提供了两种状态：一种是选中，另一种是未选中。用户通过单击该组件切换状态。

6. 单选按钮

使用 JRadioButton 创建单选按钮，为用户提供单项选择。单选按钮和复选框很相似，所不同的是，用户只能在一组单选按钮中选中一个，而且必须单击同组中其他单选按钮来切换当前单选按钮的状态。在创建了若干单选按钮之后，应使用 ButtonGroup 创建一个对象，然后利用这个对象把若干个单选按钮归组。例如：

```
ButtonGroup fruit=new ButtonGroup();
JRadioButton button1=new JRadioButton("香蕉");
JRadioButton button2=new JRadioButton("苹果");
JRadioButton button3=new JRadioButton("梨");
fruit.add(button1);
fruit.add(button2);
fruit.add(button3);
```

7. 下拉列表

使用 JComboBox 类创建下拉列表，为用户提供单项选择。用户可以在下拉列表中看到第一个选项和它旁边的箭头按钮，当用户单击箭头按钮时，选项列表打开。

8. 密码框

使用 JPasswordField 类创建密码框，允许用户在密码框中输入单行密码，密码框的默认回显字符是'*'。密码框可以使用 setEchoChar(char c)重新设置回显字符。密码框调用 char [] getPassword()方法可以返回用户在密码框中输入的密码。

6.6　事件监听器和内部类

本章前面的内容主要介绍了 Java 提供的有关图形用户界面的处理技术，包括容器、布局管理器与组件等，这些内容只涉及如何显示应用程序的用户界面，而没有涉及如何响应用户对组件的操作。

Java 采用事件处理机制响应用户的操作请求，即程序的运行过程是不断地响应各种事件的过程，事件的产生顺序决定了程序的执行顺序，事件处理是图形用户界面应用程序的重要组成部分，是实现各种操作功能的重要途径。

6.6.1　事件处理模式

1. 事件源

能够产生事件的对象都可以称为事件源，如文本框、按钮、下拉列表等。也就是说事件源必须是一个对象，而且这个对象必须是 Java 认为能够发生事件的对象。

2. 监视器

需要一个对象对事件源进行监视，以便对发生的事件作出处理。事件源通过调用相应的方法将某个对象注册为自己的监视器。例如，对于文本框，这个方法是：

```
addActionListener(监视器);
```

对于注册了监视器的文本框，在文本框中获得输入焦点后，如果用户按 Enter 键，Java 运行环境就会自动用 ActionEvent 类创建一个对象，即发生了 ActionEvent 事件。也就是说，事件源注册监视器后，相应的操作就会导致相应事件的发生，并通知监视器，监视器就会做出相应的处理。

3. 处理事件的接口

监视器负责处理事件源发生的事件。监视器是一个对象，为了处理事件源发生的事件，监视器这个对象会自动调用一个方法来处理事件（对象只有调用方法才能产生行为）。但是，监视器调用哪个方法呢？Java 规定：为了让监视器这个对象能对事件源发生的事件进行处理，创建该监视器对象的类必须申明实现相应的接口，即必须在类体中重写接口中所有方法，那么当事件源发生事件时，监视器就会自动调用被类重写的接口方法。

简而言之，Java 要求监视器必须与一个专用于处理事件的方法进行绑定，为了达到此目的，要求创建监视器类必须实现 Java 规定的接口，该接口中有专用于处理事件的方法。

6.6.2 事件类

Java 将事件分为两个类别：低级事件与语义事件。

低级事件是指来自键盘、鼠标与窗口操作有关的事件。例如，窗口极小化、关闭窗口、移动鼠标或敲击键盘等。

语义事件是指与组件有关的事件。例如，单击按钮、单击复选按钮、单击单选按钮、在文本域中输入文本信息、拖动滚动条等。

绝大部分与图形用户界面有关的事件类都位于 java.awt.event 包中，其中包含了各种事件类别的监听接口，在 javax.swing.event 包中定义了与 Swing 事件有关的事件类。

下面分别介绍低级事件与语义事件以及它们的监听器。

1. 低级事件

焦点事件、鼠标事件、键盘事件与窗口事件都属于低级事件，Java 为每种事件定义了一个标准类，用于封装具体事件的相关信息。表 6-1 中列出了低级事件的事件类名与事件描述。

表 6-1 低级事件的事件类名及描述

事件类名	描述
FocusEvent	这个事件类描述了在组件获得焦点或失去焦点时产生的事件
MouseEvent	这个事件类描述了用户对鼠标操作所产生的事件
KeyEvent	这个事件类描述了用户对键盘操作所产生的事件
WindowEvent	这个事件类描述了用户对窗口操作所产生的事件

为了更有效地处理各种类型的事件，Java 中的每一种事件对应一个监听接口，负责处理事件的监听器必须实现对应的监听器接口。图 6-5 中给出了低级事件的 5 种监听器接口。





[illegible]

2. [illegible]

[illegible]

[illegible]

[illegible]	[illegible]
ActionEvent	[illegible]
ItemEvent	[illegible]
ComponentEvent	[illegible]
ContainerEvent	[illegible]
TextEvent	[illegible]

[illegible]



[illegible]

[illegible] 内部类

[illegible]

(1) [illegible]

(2) [illegible]

(3) [illegible]

[illegible]

```
new SuperType(construction parameters)
{
    innner class methods and data
}
```

上面讲了很多关于事件以及监听器的内容，相信读者一定想知道，如何给一个事件注册监听器呢？下面将以给鼠标事件注册监听器为例来讲解这部分内容。

以匿名内部类的形式定义一个鼠标事件监听器类：

```
addMouseListener(new MouseAdapter(){
    public void mousePressed(MouseEvent event){
        ......
    }
});
```

6.7 图书管理系统 V6.0

6.7.1 运行效果图

本章我们终于给图书管理系统穿上了漂亮的"衣服"，图 6-7 便是拥有漂亮界面的图书管理系统的运行效果图。

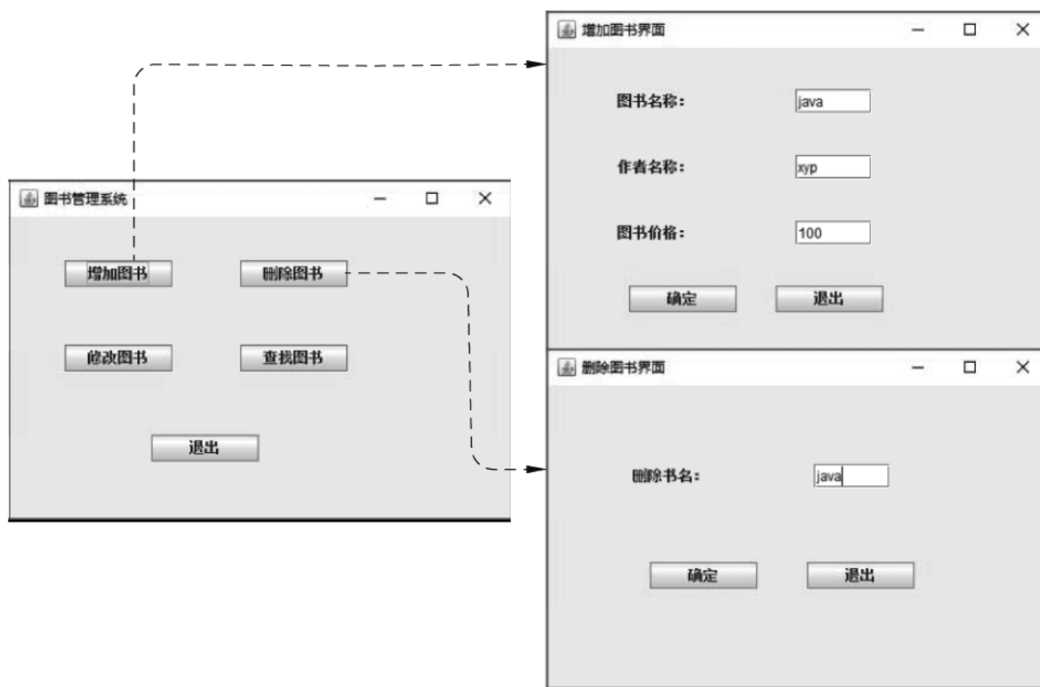


图 6-7 运行效果图

6.7.2 类结构示意图

图 6-8 和图 6-9 分别展示的是类的结构图和类的 MVC 三层结构图。

1. 类结构

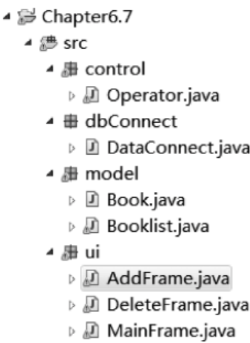


图 6-8　类结构图

2. 类 MVC 三层结构

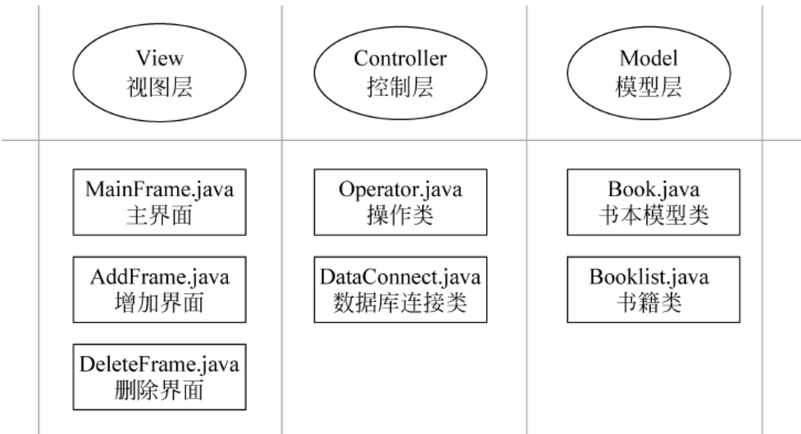


图 6-9　类 MVC 三层结构图

6.7.3 代码实现

1. 视图层代码

MainFrame.java

```
package ui;

import java.awt.BorderLayout;
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;

```

```java
import javax.swing.border.EmptyBorder;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.MouseListener;
import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;
import java.awt.event.ActionEvent;

public class MainFrame extends JFrame {

    protected static final MouseListener MainFrame = null;
    private JPanel contentPane;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    MainFrame frame = new MainFrame();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public MainFrame() {
        setTitle("图书管理系统");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JButton btnNewButton = new JButton("增加图书");
        btnNewButton.addActionListener(new ActionListener() {
```

```
            public void actionPerformed(ActionEvent e) {
                AddFrame addbook = new AddFrame();
                addbook.setVisible(true);
                MainFrame.this.dispose();
            }
        });
        btnNewButton.setBounds(47, 38, 93, 23);
        contentPane.add(btnNewButton);

        JButton btnNewButton_1 = new JButton("删除图书");
        btnNewButton_1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                DeleteFrame deletebook = new DeleteFrame();
                deletebook.setVisible(true);
                MainFrame.this.dispose();

            }
        });
        btnNewButton_1.setBounds(199, 38, 93, 23);
        contentPane.add(btnNewButton_1);

        JButton btnNewButton_2 = new JButton("修改图书");
        btnNewButton_2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
               // UpdateFrame updatebook = new UpdateFrame();
               // updatebook.setVisible(true);
               // MainFrame.this.dispose();
            }
        });
        btnNewButton_2.setBounds(47, 111, 93, 23);
        contentPane.add(btnNewButton_2);

        JButton btnNewButton_3 = new JButton("查找图书");
        btnNewButton_3.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

            }
        });
        btnNewButton_3.setBounds(199, 111, 93, 23);
        contentPane.add(btnNewButton_3);

        JButton btnNewButton_4 = new JButton("退出");
        btnNewButton_4.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
```

```
                        MainFrame.this.dispose();
                    }
                });

                btnNewButton_4.setBounds(122, 189, 93, 23);
                contentPane.add(btnNewButton_4);
        }
}
```

AddFrame.java

```
package ui;

import java.awt.BorderLayout;
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

import control.Operator;

import model.Book;

import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class AddFrame extends JFrame {

        private JPanel contentPane;
        private JTextField AddnametextField;
        private JTextField AddauthortextField;
        private JTextField AddpricetextField;

        /**
         * Launch the application.
         */
        public static void main(String[] args) {
                EventQueue.invokeLater(new Runnable() {
                        public void run() {
                                try {
                                        AddFrame frame = new AddFrame();
```

```
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public AddFrame() {
        setTitle("增加图书界面");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JLabel label = new JLabel("图书名称:");
        label.setBounds(60, 39, 95, 15);
        contentPane.add(label);

        AddnametextField = new JTextField();
        AddnametextField.setBounds(214, 36, 66, 21);
        contentPane.add(AddnametextField);
        AddnametextField.setColumns(10);

        JLabel lblNewLabel = new JLabel("作者名称:");
        lblNewLabel.setBounds(60, 96, 95, 15);
        contentPane.add(lblNewLabel);

        AddauthortextField = new JTextField();
        AddauthortextField.setBounds(214, 93, 66, 21);
        contentPane.add(AddauthortextField);
        AddauthortextField.setColumns(10);

        JLabel lblNewLabel_1 = new JLabel("图书价格:");
        lblNewLabel_1.setBounds(60, 153, 95, 15);
        contentPane.add(lblNewLabel_1);

        AddpricetextField = new JTextField();
        AddpricetextField.setBounds(214, 150, 66, 21);
```




```
            contentPane.add(AddpricetextField);
            AddpricetextField.setColumns(10);

            JButton btnNewButton = new JButton("确定");
            btnNewButton.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    String bookname = AddnametextField.getText();
                    String author = AddauthortextField.getText();
                    String Oprice = AddpricetextField.getText();
                    int price = Integer.parseInt(Oprice);
                    Book book = new Book(bookname,author,price);
                    Operator operator = new Operator();
                    boolean isSuccess = operator.addBook(book);
                    if(isSuccess)
                    {
                        JOptionPane.showMessageDialog(null, "添加图书成功!");

                    }
                    else
                    {
                        JOptionPane.showMessageDialog(null, "添加图书失败!");
                    }

                }
            });
            btnNewButton.setBounds(70, 206, 93, 23);
            contentPane.add(btnNewButton);

            JButton btnNewButton_1 = new JButton("退出");
            btnNewButton_1.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent arg0) {
                    MainFrame frame = new MainFrame();
                    frame.setVisible(true);
                    AddFrame.this.dispose();

                }
            });
            btnNewButton_1.setBounds(197, 206, 93, 23);
            contentPane.add(btnNewButton_1);
        }
}
```

DeleteFrame.java

```
package ui;

```

```java
import java.awt.BorderLayout;
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import control.Operator;

import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.ActionEvent;

public class DeleteFrame extends JFrame {

    private JPanel contentPane;
    private JTextField DeletetextField;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    DeleteFrame frame = new DeleteFrame();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public DeleteFrame() {
        setTitle("删除图书界面");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 300);
```

```
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JLabel lblNewLabel = new JLabel("删除书名:");
        lblNewLabel.setBounds(73, 71, 108, 15);
        contentPane.add(lblNewLabel);

        DeletetextField = new JTextField();
        DeletetextField.setBounds(230, 68, 66, 21);
        contentPane.add(DeletetextField);
        DeletetextField.setColumns(10);

        JButton btnNewButton = new JButton("确定");
        btnNewButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String bookname = DeletetextField.getText();
                System.out.println(bookname+"!");
                Operator operator = new Operator();
                boolean isSuccess = operator.deletebook(bookname);
                if(isSuccess)
                {
                    JOptionPane.showMessageDialog(null, "删除图书成功!");
                }
                else
                {
                    JOptionPane.showMessageDialog(null, "删除图书失败!");
                }
            }
        });
        btnNewButton.setBounds(88, 153, 93, 23);
        contentPane.add(btnNewButton);

        JButton btnNewButton_1 = new JButton("退出");
        btnNewButton_1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                MainFrame frame = new MainFrame();
                frame.setVisible(true);
                DeleteFrame.this.dispose();

            }
        });
        btnNewButton_1.setBounds(224, 153, 93, 23);
```

```
            contentPane.add(btnNewButton_1);
        }

}
```

2. 控制层代码

Operator.java

```
package control;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import dbConnect.DataConnect;
import model.Book;
import model.Booklist;

public class Operator {



        public boolean addBook(Book book)
        {
            Connection conn = DataConnect.getConnection();
            Statement s;

            try {
                s = conn.createStatement();
                String sql = "insert into booklist (bookname,author,price) values
                ('"+book.bookname+"','"+book.author+"',"+book.price+")";
                boolean isSuccess = s.execute(sql);
                return isSuccess;
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();

            }
            return false;
        }
        public boolean deletebook(String bookname)
        {
            Connection conn = DataConnect.getConnection();
            try {
```




```

            Statement s = conn.createStatement();
            String sql = "delete from booklist where bookname = '"+bookname+"'";
            boolean isSuccess = s.execute(sql);
        } catch (SQLException e) {
             // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return false;

    }
    public boolean updatename(String bookname1,String bookname2)
    {
        Connection conn = DataConnect.getConnection();
        try {
            Statement s = conn.createStatement();
            String sql = "update booklist set bookname='"+bookname2+"'where bookname='"+
            bookname1+"' ";
            boolean isSuccess = s.execute(sql);
        } catch (SQLException e) {
             // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return false;
    }
    public boolean updateauthor(String author1, String author2)
    {

        Connection conn = DataConnect.getConnection();
        try {
            Statement s = conn.createStatement();
            String sql = "update booklist set author='"+author2+"'where
            author='"+author1+"' ";
            boolean isSuccess = s.execute(sql);
        } catch (SQLException e) {
             // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return false;
    }
    public boolean updateprice(float price1, float price2)
    {
        Connection conn = DataConnect.getConnection();
        try {
```

```
                Statement s = conn.createStatement();
                String sql = "update booklist set price='"+price2+"'where price='"+price1+"' ";
                boolean isSuccess = s.execute(sql);
            } catch (SQLException e) {
                 // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return false;

        }
        public Booklist searchAll()
        {
            Booklist booklist = new Booklist();
            Connection conn = DataConnect.getConnection();
            Statement s;
            try {
                s = conn.createStatement();
                String sql = "select * from booklist ";
                ResultSet rs = s.executeQuery(sql);
                while (rs.next())
                {
                    String bookname = rs.getString(1);
                    String author = rs.getString(2);
                    int price = rs.getInt(3);
                    Book book = new Book(bookname,author,price);
                    booklist.add(book);
                }
            } catch (SQLException e) {
                 // TODO Auto-generated catch block
                e.printStackTrace();

            }
            return booklist;


        }

        public Booklist searchname(String bookname)
        {
                    Booklist booklist = new Booklist();
                    Connection conn = DataConnect.getConnection();
                    Statement s;
                    try {
```




```
                s = conn.createStatement();
                String sql = "select * from booklist where bookname = '"+bookname+"'";
                ResultSet rs = s.executeQuery(sql);
                while (rs.next())
                {
                    String bookname1 = rs.getString(1);
                    String author = rs.getString(2);
                    int price = rs.getInt(3);
                    Book book = new Book(bookname1,author,price);
                    booklist.add(book);
                }
                return booklist;
            } catch (SQLException e) {
                 // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return null;


    }
    public Booklist searchauthor(String author)
    {
        Booklist booklist = new Booklist();
        Connection conn = DataConnect.getConnection();
        Statement s;
        try {
            s = conn.createStatement();
            String sql = "select * from booklist where author = '"+author+"'";
            ResultSet rs = s.executeQuery(sql);
            while (rs.next())
            {
                String bookname = rs.getString(1);
                String author1 = rs.getString(2);
                int price = rs.getInt(3);
                Book book = new Book(bookname,author1,price);
                booklist.add(book);

            }
            return booklist;
        } catch (SQLException e) {
             // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

```
            return null;


        }
        public Booklist searchprice(float price)
        {
            Booklist booklist = new Booklist();
            Connection conn = DataConnect.getConnection();
            Statement s;
            try {
                s = conn.createStatement();
                String sql = "select * from booklist where price = 'price'";
                ResultSet rs = s.executeQuery(sql);
                while (rs.next())
                {
                    String bookname = rs.getString(1);
                    String author = rs.getString(2);
                    int price1 = rs.getInt(3);
                    Book book = new Book(bookname,author,price1);
                    booklist.add(book);

                }
                return booklist;
            } catch (SQLException e) {
                 // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return null;


        }
        public static void main(String[] args) {
             // TODO Auto-generated method stub

        }

    }
```

DataConnect.java

```
package dbConnect;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```java
import java.sql.Statement;

public class DataConnect {


        static final String url = "jdbc:mysql://localhost:3306/book?characterEncoding=utf8&useSSL=true";
        static final String name = "com.mysql.jdbc.Driver";
        static final String user = "root";
        static final String password = "mysql123";
         //static Connection con = null;
         //static Statement statement= null;

        static
        {
            try
                {
                    Class.forName(name);          //指定连接类型
                    System.out.println("lianije");
                }
            catch(ClassNotFoundException cE)
                {
                    System.out.println("Class Not Found Exception:"+cE.toString());
                }
        }
        public static Connection getConnection()
        {
            try {
                return DriverManager.getConnection(url, user, password);
                } catch (SQLException e) {
                e.printStackTrace();
                return null;
            }
        }
        public static void closeConnection(ResultSet rs,Statement statement,Connection con)
        {
                try {
                    if(rs!=null)rs.close();
                    if(statement!=null)statement.close();
                    if(con!=null)con.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
        }
```

```
        public static void closeConnection(Statement statement,Connection con)
        {
            closeConnection(null,statement,con);
        }
}
```

3. 模型层代码

Book.java

```
package model;

public class Book {
    public String bookname;

    public String author;

    public int price;

    public Book(String bookname, String author, int bookPrice)
    {
        this.bookname = bookname;
        this.author = author;
        this.price = bookPrice;
    }

}

```

Booklist.java

```
package model;

import java.util.Vector;

public class Booklist extends Vector<Book> {
}
```

6.8 把我们的界面变漂亮

在学习完本章之后，相信读者已经对 GUI 编程有了一定的认识，我们已经将白底黑字的图书管理系统变成了拥有界面的图书管理系统，但是怎么将图书管理系统继续升级（使界面更加漂亮）呢？笔者想要给大家介绍一种 Java Swing 提供的 LookAndFeel（皮肤）机制。

从功能上来说，LookAndFeel 是一种批量管理 Swing 控件外观的机制；从根源上来说，LookAndFeel 是 Swing 的核心。众所周知，Swing 优于 AWT 有个很重要的原因就是

Swing 支持跨平台。在 Java 中,为了让用户能够动态地更改应用的外观,从而给用户更好的体验,提供了很多的 LookAndFeel,可以适应不同的操作系统。使用 LookAndFeel 的方式非常简单,只需在 main 函数中加入 UIManager. setLookAndFeel(String s)一行代码,即可改变外观。

除了以上的基本用法以外,还有很多有关 LookAndFeel 开源的一些套件,如:Weblaf、PgsLookAndFeel、Seaglass、beautyeye 等,建议读者可以自己多了解,然后将图书管理系统的界面变得更加美丽。

第 7 章

网 络 编 程

从本章起，我们将讲解关于 Java 网络编程的相关知识，让我们的程序互连互通，不同的技术带给我们截然不同的程序功能和效果。同时，如果在本章基础上稍作改动，即可做出一个简易版的 QQ 程序。

7.1 本 章 任 务

理论任务：理解网络编程中几个重要问题，理解 TCP、UDP 和 HTTP 编程原理，重点掌握 TCP 编程。

实践任务：将我们的图书管理系统改造成客户端-服务器版的程序，即基于 TCP 的 C/S 版图书管理系统。

7.2 网络的几个重要问题

根据以往的教学经验，在网络编程之前，大家需要再复习一下计算机网络与网络编程的几个重要问题：(1)网络为什么要分层？(2)每一层大致都做了什么事？(3)什么是协议？它们由哪些人制定？协议落实的实体是什么？(4)计算机网络的本质特点是什么？为什么叫计算机网络而不叫计算机链路或计算机传输？传统的电话聊天和 QQ/微信语音聊天的本质区别是什么？(5)Java 里如何调用这些协议的接口？(6)常见的 TCP/UDP 编程和 HTTP 编程都是怎么回事？

首先我们来简单地说一下网络为什么要分层这个问题。当一个问题比较复杂的时候，常见处理方式有两种：一是横向分解，比如下课后我们要开个班会，大家高兴一下。张三同学带着几个同学布置场地，李四同学和几个同学去买吃的，等等。这样的协作形式就是横向的。这种横向分工的特点是大家互不打搅。李四没有买好吃的不会影响张三等同学布置场地。落实到软件领域，比如做一个简单的网站或 BBS，各个模块/板块之间没有耦合或耦合很少，除了访问同一个数据库，其他几乎没有关系，这种系统相对较简单，很多同学都有搭建过 BBS 的经历。二是纵向分解，当问题更复杂的时候，往往就没这么简单了，比如我们在淘宝上网购，下订单—商家发货—物流—买家收货—淘宝确认等，这个过程中的每一步是承前启后的、彼此连贯的，这就是分层。计算机网络就是一个非常庞大的软硬件系统，要让计算机能够通过连接的方式互连互通，可真是一个大工程，这样大的工程就只有纵向地分层了。这里就不再扩展了。

接着，我们来讲述计算机网络每一层的内涵。当然，这里只是简单地总结一下，不可能像计算机网络课程里那样，讲上一学期。不过，说实话，我当年上大学学习计算机网络的时候，从第一节课就听老师说网络要分层……ISO/OSI 啥的，课程结束时也只知道要分层，其他的真不懂。想搭个局域网，不会；想写个 QQ，不会；做个网站，更是离得太远。心里又虚又着急，一直在纠结：连这都不会还学啥计算机呀。从我这些年的教学来看，很多同学和我当年读书时一个水平。这也是我在上这门课时为什么要一开始带着同学们复习的原因。如果有的同学已经懂了，那可以忽略本节的内容。下面，我想用一句话分别总结计算机网络物理层、数据链路层、网络层、传输层、应用层的功能和作用。

物理层：相邻节点数据的传输。这里，相邻节点指的是网络中同一网段的主机。做个比喻：同学们的寝室，一个楼层的寝室，中间是走廊，两边有很多房间。我在大学时的房间号是 6405，我隔壁是 6403，再隔壁是 6401，我的对面是 6406。如果把每个寝室看成节点，整个一层楼的走廊是总线，那么，所有 64** 房间都是我这里所说的相邻节点。比如我在 6405 大喊："张三，吃饭去。"而几乎同时，隔壁李四也在喊："王五，踢球去。"物理层就是只管传，不管传到没有，传好没有。

数据链路层：相邻节点数据可靠的传输。就像上面例子里那样，我在 6405 大喊："张三，吃饭去。"而几乎同时，隔壁李四也在喊："王五，踢球去。"如果这两个声音几乎同时发生，在我们人类世界是没事的，因为我们太智能了，可以过滤掉我们不需要的信息。但是，计算机不一样。这种事情发生在物理层就会产生信号的叠加，就会产生错误。因此，链路层就是要纠错，保证相邻节点数据"可靠地"传输。所以才有大家计算机网络课上学习的链路层滑动窗口等协议。

网络层：源端到目标端数据的传输。还是以寝室里互相喊话的例子来说，既然链路层解决了同一层寝室同学互相喊话的问题，那边网络层要解决的是同一栋楼不同层寝室之间、不同栋楼寝室之间互相喊话的问题。因为同一栋楼的同一层里寝室间通信直接喊就行，但不同楼层、不同楼宇之间就得有"路由器"来转告了。网络层就是要解决当数据传输跨网络、跨网段时，源端到目标端数据的传输。注意：计算机网络最主要的特点都在网络层。同学们知道，计算机网络中除了端系统外，在网络中最主要的设备就是路由器，而路由器就在网络层工作的设备中，没有传输层和应用层。

讲到这里，自然要解释一个问题：计算机网络的本质特点是什么？为什么叫计算机网络而不叫计算机链路或计算机传输？传统的电话聊天和 QQ/微信语音聊天的本质区别是什么？

计算机网络的本质特点就是网络层的本质特点：分组交换、无连接、尽力而为、不可靠。这几个关键词希望同学们一定要理解、记忆。下面分别简单讲解一下：首先，分组交换这一点是相对于电路交换来说的。想必大家在计算机网络这门课程中学过，传统的通信网络、电话网络是电路交换的，电话从拨号到连接到放回铃音到用户接通电话后，一个完整的电路由电话的两位用户"独享"。这种独享的电路交换服务可谓是 VIP 服务，特别是在早期，和网络电话相比，传统电话服务质量非常好。分组交换的特点就像卡车运货，同一批货物，不同卡车可以走不同路径，最后运到目的地即可。这种分组交换的好处是不独享链路，整体链路利用率高。但是也有不足，服务质量没有电路交换好，因为常丢包、常

重传，卡车运的货到了目的地还有重新组装。不过，随着现代技术的快速发展，现在的计算机网络服务质量越来越好，所以大家现在可以用 QQ/微信等这样的计算机网络工具打电话和视频了。当前的通信网络，包括目前的 5G 网络，都在使用分组交换的方式了。

计算机网络还有一个特点，就是无连接、尽力而为、不可靠。既然是分组交换，每个分组包在由网络上路由器传递到下一个路由器时，要注意，在这个过程中，路由器之间是无连接的、尽力而为的、不可靠的。大家都知道，在计算机网络中，除了终端以外，在网络中，只有路由器，最高就是网络层，因此计算机网络最本质的特点就是分组交换、无连接、尽力而为、不可靠。

传输层：源端到目标端数据可靠的传输。既然网络层的本质特点是分组交换、无连接、尽力而为、不可靠，那么如果我们希望得到可靠的、有连接的网络服务该怎么办？举个例子，比如我是一位老板，我雇了一些初级秘书，现在我打算让张三到我办公室来一下，于是我告诉了我办公室的一个初级秘书，让这个初级秘书去喊张三到我办公室来。这里的初级秘书就是网络层，他们都是尽力而为的、不负责的，只是尽力把这个消息传给下一个初级秘书，直到找到张三。这就是网络层干的事。大家知道，在这个过程中很有可能由于网络丢包、拥塞等没有找到张三，这样的服务老板肯定受不了。于是，我们在不改变网络层，也就是在不辞退这个初级秘书的前提下，在每个终端用户的办公室都雇了一个高级秘书（注意：这个高级秘书只在终端设备上有，在网络内部还是那些尽力而为的初级秘书），这样，老板有什么事只需要和这个高级秘书说一下，这位高级秘书来通知那些初级秘书，他通过重传、握手等机制保证向老板提供一种感觉上可靠的、面向连接的服务——这就是 TCP。同学们注意，TCP 是给用户提供了一种感觉上可靠的、面向连接的服务。而网络的本质并没有变，分组交换、无连接、尽力而为、不可靠。这里再举个例子，拿快递公司来做比喻，路上我们见到的快递小哥就类似刚才的初级秘书，他们可能因为堵车、抛锚等各种原因无法把货物及时送到，但是自从有了快递公司以后，用户就不担心了，因为快递公司给我们提供服务，让我们感觉他是安全的、可靠的，虽然网络中实际送货的还是快递小哥。

当然，并不是所有人都需要可靠的服务，因为可能有时需要代价，比如重传机制就需要用户为了可靠而等待一下。但是，有些应用，比如在线看直播，就需要实时，而不是为了可靠反复重传，结果比赛结束了。所以在传输层还有另外一个很重要的协议——UDP。总的来说，不同应用需求选择不同传输协议。

应用层：应用层无法用一句话总结，因为应用太多，千差万别。简单点说，绝大多数应用在传输层都需要使用 TCP 或 UDP，但是这绝大多数的应用又不能直接一点不改地使用 TCP 或 UDP。比如经常用来传输网页的 HTTP 协议，它急需要在传输层用 TCP，又不能简简单单地直接使用 TCP。为什么？因为 TCP 是长连接，连上就不断开，对于网页浏览这个应用根本就不需要长连接，为了让服务器能够服务更多用户（当然还有其他一些原因），就有了 HTTP 协议。这是一个请求响应即断开、基于字符的、无状态的协议。所谓无状态的意思，是相对于 TCP 说的，既然 TCP 是长连接，一个 TCP 客户端连上了服务器，告诉服务器我是谁，那么服务器一直都知道连上的客户端是谁。但是 HTTP 不一样，它的客户端有问题要咨询服务器时才连接、发送请求，收到服务器响应就断开了，下次再连服务器，服务器又不知道这个客户端是谁了，这就是无状态的意思。这里有同学可能

要说，Web 上有 session、cookie 等解决这个问题。对，不过那是后面章节的内容，并且也是不矛盾的，只是解决这种无状态问题的办法而已，这里不作讲解。

上面，我们常提到一个词：协议。那么，什么是协议呢？谁来定义协议呢？定义好的协议落实的实体是什么？Java 怎么调用这个协议呢？这里，我们逐一解释一下。

协议，就是一组规则。比如，每个学生入校时，学校都要给你发一个学生手册，那上面有很多规则；又如一个大学生在大学期间要是挂了 * 门课，就得留级；要是挂了 ** 门课，就得退学，等等。那一本书都是写规则。再举个打麻将的例子，比如大家都知道有成都麻将和重庆麻将两种麻将。成都麻将又叫血战到底，重庆麻将又叫倒倒胡。这是两种不同的游戏规则，如果四个人坐到一起，其中三个人要玩成都麻将，而另一个人只会玩重庆麻将，那你说会怎么样？结果就是这个人无法玩，落实在计算机网络里就是这个人无法和其他人互连互通。再比如我们要举办一个国际麻将大会，为了公平公正，我们一开始就要公布全部完整的游戏规则。不管是学校的学生手册还是打麻将的例子，最后这些协议、这些规则，落实的实体无非就两样：(1)一份白底黑字的详细的说明规则纸质材料；(2)把这份规则写成代码，实现协议。

那么谁来定义计算机网络中的这些协议呢？有那么一大帮人，他们有技术专家、有各大财团代表、有国家代表、有公司代表，也有跑龙套的、牵线搭桥的，这些人组织在一起，经常成立一些组织，做大了就叫国际组织。在计算机网络里，有个叫 IETF 的组织，这是个国际性的重要组织，它们定义了互联网上绝大多数规则、协议，并且每个协议都有一个编号代表，以 RFC *** 这样的形式命名。比如我们这门课要学习的 HTTP，百度“HTTP RFC”，搜索结果第一条就是 HTTP 对应的那份白底黑字的材料。这些人为什么要积极地主导或者参与定义这些协议、标准呢？标准的制定，对于知识产权、企业利益、国家战略等都是非常重要的。

有人定义就要有人实现，既然 IETF 把这些协议定义了，接下来就有很多企业、高校、研究机构把这些协议变成代码实现，比如在计算机网络里大家都知道的企业像华为、思科等。作为应用程序开发人员，我们并不关心我的电脑里的协议都是哪个企业实现的，他们都是按照协议标准实现，提供给我们的接口调用都是一样的。

在这些标准接口调用中，其中涉及 TCP 网络编程的接口，我们称之为 Socket 编程，在 Java 里涉及 TCP 编程的部分我们称之为 Java Socket 编程。同时，Java 也提供了 UDP 编程和 HTTP 编程的类库。

到此为止，我们把在教学网络中的几个重要问题简单地梳理了一下，接下来进行具体分析。

7.3 TCP 编程

7.3.1 什么是 TCP/IP

TCP/IP 是 Transmission Control Protocol/Internet Protocol 的简写，译名为传输控制协议/因特网协议，是 Internet 最基本的协议。TCP/IP 是这个协议族的统称，它采用

了5层的层级结构,分别是物理层、数据链路层、网络层、传输层、应用层。上一节已经对各个层级的功能和作用进行了介绍。下面主要介绍TCP建立连接的步骤,分为阻塞式和非阻塞式。

7.3.2 TCP建立连接步骤(阻塞式)

TCP提供面向连接的服务,通过它建立的是可靠的连接。Java为TCP协议提供了两个类:Socket类和ServerSocket类。一个Socket实例代表了TCP连接的一个客户端,而一个ServerSocket实例代表了TCP连接的一个服务器端。一般在TCP Socket编程中,客户端有多个,而服务器端只有一个。客户端TCP向服务器端TCP发送连接请求,服务器端的ServerSocket实例则监听来自客户端的TCP连接请求,并为每个请求创建新的Socket实例,由于服务端在调用accept()等待客户端的连接请求时会阻塞,直到收到客户端发送的连接请求才会继续往下执行代码,因此要为每个Socket连接开启一个线程。服务器端要同时处理ServerSocket实例和Socket实例,而客户端只需要使用Socket实例。另外,每个Socket实例会关联一个InputStream和OutputStream对象,通过将字节写入套接字的OutputStream来发送数据,并通过InputStream接收数据。

客户端向服务器端发送连接请求后,就被动地等待服务器的响应。典型的TCP客户端要经过以下三步操作:

(1) 创建一个Socket实例:构造函数向指定的远程主机和端口建立一个TCP连接;

(2) 通过套接字的I/O流与服务端通信;

(3) 使用Socket类的close方法关闭连接。

服务端的工作是建立一个通信终端,并被动地等待客户端的连接。典型的TCP服务端执行如下两步操作:

(1) 创建一个ServerSocket实例并指定本地端口,用来监听客户端在该端口发送的TCP连接请求;

(2) 重复执行:调用ServerSocket的accept()方法以获取客户端连接,并通过其返回值创建一个Socket实例;为返回的Socket实例开启新的线程,并使用返回的Socket实例的I/O流与客户端通信;通信完成后,使用Socket类的close()方法关闭该客户端的套接字连接。

7.3.3 TCP建立连接步骤(非阻塞式)

以上我们介绍了建立阻塞式TCP连接的方法和步骤。但当我们想采用非阻塞的方式时,就需要采取基于NIO的方式进行TCP通信。那么,什么是NIO?如何进行非阻塞式TCP编程呢?

NIO采取通道(Channel)和缓冲区(Buffer)来传输和保存数据,它是非阻塞式的I/O,即在等待连接、读写数据的时候,程序也可以做其他事情,以实现线程的异步操作。

考虑一个即时消息服务器,可能有上千个客户端同时连接到服务器,但是在任何时刻只有非常少量的消息需要读取和分发(如果采用线程池或者一线程一客户端方式,则会非常浪费资源),这就需要一种方法能阻塞等待,直到有一个信道可以进行I/O操作。NIO

的 Selector 选择器就实现了这样的功能，一个 Selector 实例可以同时检查一组信道的 I/O 状态，它就类似一个观察者，只要我们把需要探知的 SocketChannel 告诉 Selector，我们接着做别的事情。当有事件(比如连接打开、数据到达等)发生时，它会通知我们，传回一组 SelectionKey，读取这些 Key，就会获得我们刚刚注册过的 SocketChannel，然后，我们从这个 Channel 中读取数据，接着可以处理这些数据。

Selector 的内部原理实际是在做一个对所注册的 Channel 的轮询访问，不断地轮询(目前就这一个算法)，一旦轮询到一个 Channel 有所注册的事情发生，比如数据来了，它就会读取 Channel 中的数据，并对其进行处理。

要使用选择器，需要创建一个 Selector 实例，并将其注册到想要监控的信道上(通过 Channel 的方法实现)。最后调用选择器的 select()方法，该方法会阻塞等待，直到有一个或多个信道准备好了 I/O 操作或等待超时，或另一个线程调用了该选择器的 wakeup()方法。现在，在一个单独的线程中，通过调用 select()方法，就能检查多个信道是否准备好进行 I/O 操作，由于非阻塞 I/O 的异步特性，在检查的同时，我们也可以执行其他任务。

基于 NIO 的 TCP 连接的建立步骤如下。

1. 服务端

(1) 传建一个 Selector 实例；

(2) 将其注册到各种信道，并指定每个信道上感兴趣的 I/O 操作；

(3) 重复执行：调用一种 select()方法；获取选取的键列表；对于已选键集中的每个键，获取信道，并从键中获取附件(如果为信道及其相关的 Key 添加了附件的话)；确定准备就绪的操作并执行，如果是 accept 操作，将接收的信道设置为非阻塞模式，并注册到选择器。

2. 客户端

与基于多线程的 TCP 客户端大致相同，只是这里是通过信道建立连接，但在等待连接建立及读写时，我们可以异步地执行其他任务。

7.4 UDP 编程

7.4.1 什么是 UDP

UDP 提供的服务不同于 TCP 的端到端服务，它是面向非连接的，属不可靠协议，UDP 套接字在使用前不需要进行连接。实际上，UDP 协议只实现了两个功能：

(1) 在 IP 协议的基础上添加了端口；

(2) 对传输过程中可能产生的数据错误进行了检测，并抛弃已经损坏的数据。

Java 通过 DatagramPacket 类和 DatagramSocket 类来使用 UDP 套接字，客户端和服务器端都通过 DatagramSocket 的 send()方法和 receive()方法来发送和接收数据，用 DatagramPacket 来包装需要发送或者接收到的数据。发送信息时，Java 创建一个包含待发送信息的 DatagramPacket 实例，并将其作为参数传递给 DatagramSocket 实例的 send()方法；接收信息时，Java 程序首先创建一个 DatagramPacket 实例，该实例预先分

配了一些空间,并将接收到的信息存放在该空间中,然后把该实例作为参数传递给DatagramSocket实例的receive()方法。在创建DatagramPacket实例时,要注意:如果该实例用来包装待接收的数据,则不指定数据来源的远程主机和端口,只需指定一个缓存数据的byte数组即可(在调用receive()方法接收到数据后,源地址和端口等信息会自动包含在DatagramPacket实例中),而如果该实例用来包装待发送的数据,则要指定要发送到的目的主机和端口。

由上我们可以知道UDP是无连接的、不可靠协议。当我们使用UDP协议实现简易版QQ的群聊时,会出现有的客户端接收数据不全或者接收不到数据。这里就要注意,如果要实现文件的可靠传输,就必须在上层对数据丢包作特殊处理。

7.4.2 UDP建立连接步骤

UDP客户端首先向被动等待联系的服务器发送一个数据报文。一个典型的UDP客户端要经过以下三步操作:

(1) 创建一个DatagramSocket实例,可以有选择地对本地地址和端口号进行设置,如果设置了端口号,则客户端会在该端口号上监听从服务器端发送来的数据;

(2) 使用DatagramSocket实例的send()和receive()方法来发送和接收DatagramPacket实例,进行通信;

(3) 通信完成后,调用DatagramSocket实例的close()方法来关闭该套接字。

由于UDP是无连接的,因此UDP服务端不需要等待客户端的请求以建立连接。另外,UDP服务器为所有通信使用同一套接字,这点与TCP服务器不同,TCP服务器则为每个成功返回的accept()方法创建一个新的套接字。一个典型的UDP服务端要经过以下三步操作:

(1) 创建一个DatagramSocket实例,指定本地端口号,并可以有选择地指定本地地址,此时,服务器已经准备好从任何客户端接收数据报文;

(2) 使用DatagramSocket实例的receive()方法接收一个DatagramPacket实例,当receive()方法返回时,数据报文就包含了客户端的地址,这样就知道了回复信息应该发送到什么地方;

(3) 使用DatagramSocket实例的send()方法向服务器端返回DatagramPacket实例。

7.5 HTTP编程

7.5.1 什么是HTTP,为什么要有HTTP

HTTP(Hypertext Transfer Protocol,超文本传输协议)从1990年开始就在WWW上广泛应用。HTTP是一个属于应用层的面向对象的协议,由于其简捷、快速的方式,适用于分布式超媒体信息系统。HTTP是应用层协议,当你上网浏览网页的时候,浏览器和服务器之间就会通过HTTP在Internet上进行数据的发送和接收,也是基于请求和响

应的、无状态的协议。

我们知道 HTTP 是基于 TCP 的。那么,它们之间又有什么区别呢? 区别在于:TCP 是面向连接的协议。传输连接是用来传送 TCP 报文的。TCP 传输连接的建立和释放是每一次面向连接的通信中必不可少的过程。因此,传输连接就有三个阶段,即连接建立、数据传送和连接释放。而 HTTP 本身是无连接的。这就是说,虽然 HTTP 使用了 TCP 连接,但通信的双方在交换 HTTP 报文之前不需要先建立 HTTP 连接。并且 HTTP 是无状态的。也就是说,同一个客户第二次访问同一个服务器上的页面时,服务器的响应与第一次被访问时的相同(假设现在服务器还没有把页面更新),因为服务器并不记得曾经访问过这个客户,也不记得为该客户曾经服务过多少次。HTTP 的无状态特性简化了服务器的设计,使服务器更容易支持大量并发的 HTTP 请求。

7.5.2 HTTP建立连接步骤

在 Java 中使用 HTTP 通信的客户端需要用到 java.net 包中的 HttpURLConnection 类,每个 HttpURLConnection 实例都可用于生成单个请求,请求完成后在 HttpURLConnection 的 InputStream 或 OutputStream 上调用 close()方法可以释放与此实例关联的网络资源。表 7-1 是 HttpURLConnection 的常用方法。

表 7-1 HttpURLConnection 的常用方法

方　法	功能说明
HttpURLConnection(URL u)	构造方法
String getRequestMethod(String method)	获取请求方法
int getResponseCode()	从 HTTP 响应消息获取状态码

在 HttpURLConnection 构造方法中通过 URL 对象指明要访问的 URL 地址。URL 在 java.net 包中,表 7-2 是 URL 的常用方法。

表 7-2 URL 的常用方法

方　法	功能说明
URL(String spec)	根据 String 表示形式创建 URL 对象
String getHost()	获得此 URL 的主机名
URLConnection openConnection()	返回一个 URLConnection 对象,它表示到 URL 所引用的远程对象的连接
InputStream openStream()	打开到此 URL 的连接并返回一个用于从该连接读入的 InputStream

7.6 客户/服务器模式

本节采用七个实例向读者讲解从最简单的控制台输入输出到网络上通过 Socket 建立 C/S 模式的通信。讲解 C/S 模式采用由简单到复杂的思路:一个客户端和一个服务器端进行一次通信、一个客户端和一个服务器端进行多次通信、多个客户端和一个服务器

端进行通信，最后讲解客户端和服务器端通过 HTTP 进行通信。

7.6.1　控制台上的简单输入输出

本节实现了一个在控制台上输入并且输出结果的事例，代码在工程 Chap7.6.1 中，运行本工程，在控制台中输入 hello 字符，如图 7-1 所示。

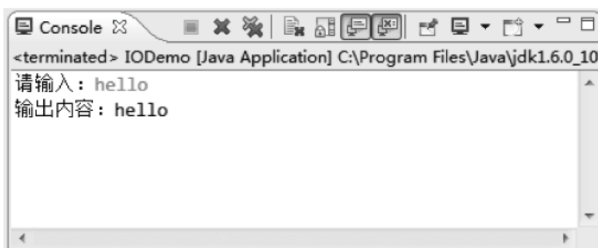


图 7-1　Chap7.6.1 执行结果

Chap7.6.1 代码如下所示：

```
1   package io;
2   import java.util.Scanner;
3
4   public class IODemo {
5
6       public IODemo() {
7           System.out.print("请输入：");
8           Scanner scanner = new Scanner(System.in);
9           String str = scanner.next();
10          System.out.println("输出内容："+str);
11      }
12
13      public static void main(String[] args) {
14          new IODemo();
15      }
16
17  }
```

在第 8 行通过 java.util 包中的 Scanner 类实现控制台的输入输出，在第 9 行通过 Scanner 类的 next()方法得到输入到控制台中的字符串，并在第 10 行通过 System.out 输出字符串到控制台。

7.6.2　控制台上的循环输入输出

本小节的工程是在控制台上的循环输入输出，在 7.6.1 工程的基础上加入了一个 while 循环，使程序可以循环输入输出，当输入 exit 字符串时，程序退出循环。本工程在 Chap7.6.2 中，运行本工程，在控制台中输入 hello 字符和 exit 字符的效果如图 7-2 所示。

工程 Chap7.6.2 代码如下所示：

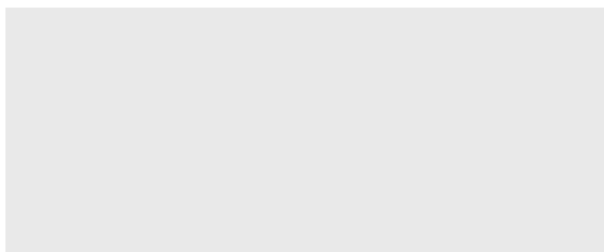


图 7-2　Chap7.6.2 执行结果

```
1   package io;
2   import java.util.Scanner;
3
4   public class LoopIODemo {
5
6       public LoopIODemo() {
7           while (true) {
8               System.out.print("请输入:");
9               Scanner scanner = new Scanner(System.in);
10              String str = scanner.next();
11              if (str.equals("exit")) {
12                  System.out.println("退出");
13                  break;
14              } else {
15                  System.out.println("输出内容:" + str);
16              }
17          }
18      }
19
20      public static void main(String[] args) {
21
22          new LoopIODemo();
23      }
24
25  }
```

本程序与 Chap7.6.1 相比在第 7 行添加了一个 while 循环实现在控制台循环输入的功能，每次在控制台上输入的数据都会在第 11 到第 13 行的条件语句中判读，如果输入的字符是 exit，则执行第 12 到第 13 行退出 while 循环终止程序。

7.6.3　一个客户端和一个服务器一次通信

从这一节起我们将进入客户端与服务器端的 Socket 通信，首先是最简单的一个客户端和一个服务器端进行一次通信。工程 Chap7.6.3 有两个包：client 包和 server 包，在 client 包中是客户端 SocketClient；在 server 包中是服务器端 SocketServer。首先运行

服务器端的 SocketServer 类,没有连接客户端时的效果如图 7-3 所示。

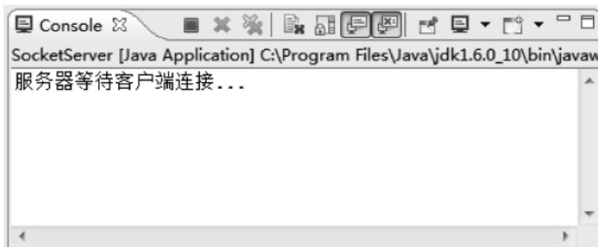


图 7-3 服务器未连接客户端时

此时没有客户端连入,所以服务器端一直等待客户端的连入。当我们运行 SocketClient 类时,服务器端的控制台运行效果如图 7-4 所示,客户端的控制台如图 7-5 所示。

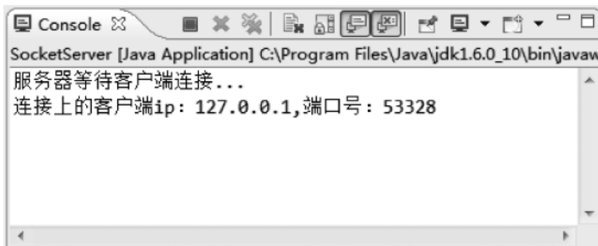


图 7-4 服务器端

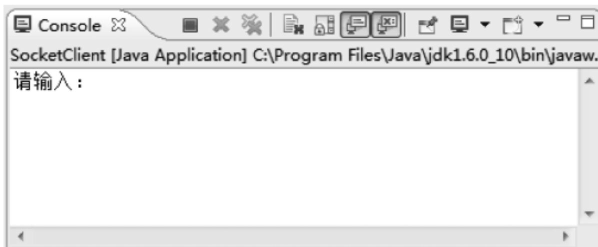


图 7-5 客户端

当在客户端的控制台中输入如图 7-6 所示的数据发送给服务器端时,服务器端接收数据并返回一个字符串,然后断开与客户端的连接,如图 7-7 所示。

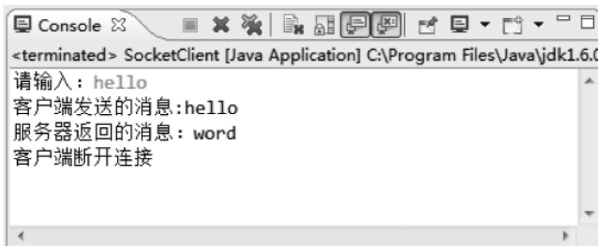


图 7-6 客户端发送消息

以下我们来讲解 Chap7.6.3 的代码,首先是客户端 SocketClient,如下所示。

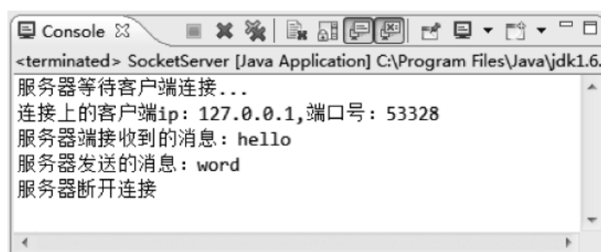


图 7-7　服务器端接受消息并返回

```
package client;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;

public class SocketClient {
    public SocketClient() {
            Socket socket = null;
            OutputStream out = null;
            InputStream in = null;
            try {
                socket = new Socket("localhost", 8008);
                out = socket.getOutputStream();
                System.out.print("请输入:");
                Scanner scanner = new Scanner(System.in);
                String mes = scanner.next();
                System.out.println("客户端发送的消息:" + mes);
                out.write(mes.getBytes());
                in = socket.getInputStream();
                byte[] buffer = new byte[1024];
                int index = in.read(buffer);
                String receive = new String(buffer, 0, index);
                System.out.println("服务器返回的消息:" + receive);
                System.out.println("客户端断开连接");
                in.close();
                out.close();
                socket.close();
            } catch (UnknownHostException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
```

```
36        }
37
38        public static void main(String[] args) {
39            new SocketClient();
40
41        }
42
43  }
```

客户端 SocketClient 在包 client 中，首先在第 15 行通过 socket 连接本地监听端口号为 8008 的服务器端，在第 16 行和第 22 行得到 socket 的输出输入流，通过得到的输出对象 out 向服务器端发送控制台中输入的消息，然后在通过输入对象 in 接收来自服务器端的返回消息，最后第 28 到第 30 行关闭各个连接。

下面我们来了解一下服务器端 SocketServer 是如何实现的，代码如下所示：

```
package server;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class SocketServer {

    public SocketServer() {
        Socket socket = null;
        OutputStream out = null;
        InputStream in = null;
        try {
            ServerSocket serverscoket = new ServerSocket(8008);
            System.out.println("服务器等待客户端连接...");
            socket = serverscoket.accept();
            String ip = socket.getLocalAddress().getHostAddress();
            int port = socket.getPort();
            System.out.println("连接上的客户端 ip:" +  ip + ",端口号:" + port);
            in = socket.getInputStream();
            byte[] buffer = new byte[1024];
            int index = in.read(buffer);
            String receive = new String(buffer, 0, index);
            System.out.println("服务器端接收到的消息:" + receive);
            out = socket.getOutputStream();
            String mes = "word";
            out.write(mes.getBytes());
            System.out.println("服务器发送的消息:" + mes);
```

```
30                System.out.println("服务器断开连接");
31                in.close();
32                out.close();
33                socket.close();
34                serverscoket.close();
35
36            } catch (IOException e) {
37                e.printStackTrace();
38            }
39
40        }
41
42        public static void main(String[] args) {
43              new SocketServer();
44        }
45
46    }
```

服务器端 SocketServer 在包 server 里，首先在第 15 行通过 ServerSocket 监听 8008 端口，通过 accept()方法接受客户端的连接，如果没有客户端连入，则程序一直阻塞在第 17 行，如图 7-3 所示。当客户端接入后，开始执行下面的代码通过 socket 获取客户端的 ip 地址和端口号，并且得到与客户端进行通信的 socket 输入输出流，完成和客户端之间的通信。当服务器端在第 24 行得到了客户端发送的消息，立即在第 28 行返回一个字符串给客户端，然后关闭各个流，完成一次客户端和服务器端的通信。

本小节讲述了客户端和服务器端的一次通信，在后面的小节中将为读者讲述客户端与服务器端的进一步交互。

7.6.4 一个客户端和一个服务器多次通信

上一小节我们讲述了一个客户端和一个服务器端的一次通信，在本小节将对上一小节的功能进行扩充，让客户端能与服务器多次通信，直到发送退出消息结束通信。Chap7.6.4 在 Chap7.6.3 上进行了改动，修改了客户端 SocketClient，在 SocketClient 中增加了一个 while 循环，使客户端可以一直发送消息给服务器端，直到客户端发送退出消息。而在服务器端的 SocketServer 中也增加了一个 while 循环用于一直读取客户端发送的消息。下面我们来了解一下 Chap7.6.4 的运行效果，首先分别运行服务器端和客户端，服务器端如图 7-8 所示。

然后在客户端的控制台输入“hello”消息发送给服务器端，客户端的控制台如图 7-9 所示。

从图 7-9 中可以看出客户端还可以继续输入消息，那么我们再输入一条消息“hi”，此时客户端的控制台如图 7-10 所示，服务器端如图 7-11 所示。

最好当客户端发送“exit”消息时告诉服务器断开连接，客户端如图 7-12 所示，服务器端如图 7-13 所示。

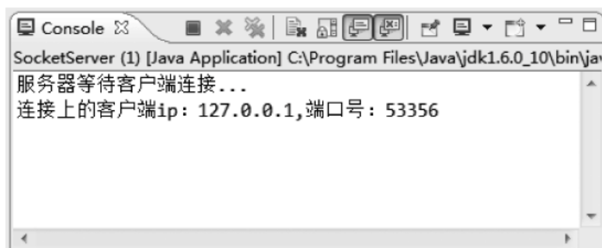


图 7-8　服务器端

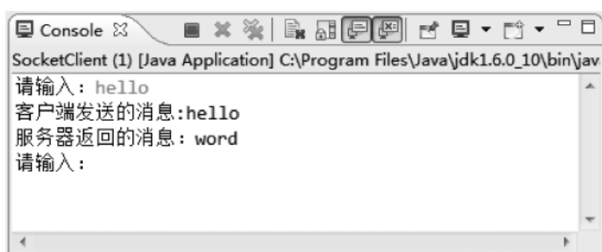


图 7-9　客户端发送一条消息

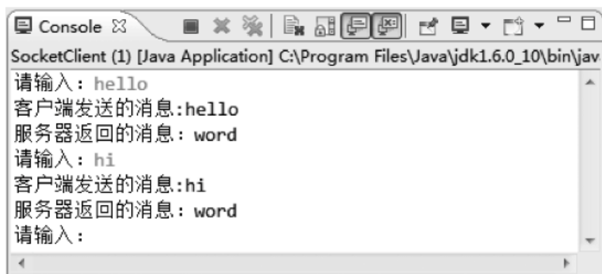


图 7-10　客户端发送第二条消息

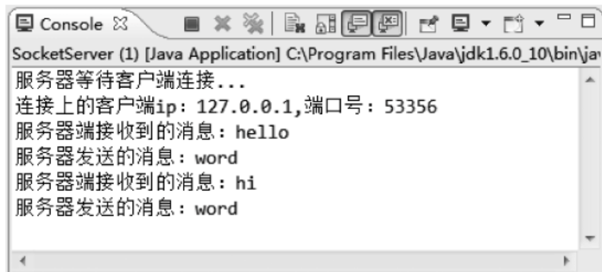


图 7-11　服务器端接收了两条消息

从以上的图中我们可以发现客户端与服务器端进行了多次通信，下面我们就来讲解如何实现此功能。首先讲解客户端 SocketClient，代码如下所示：

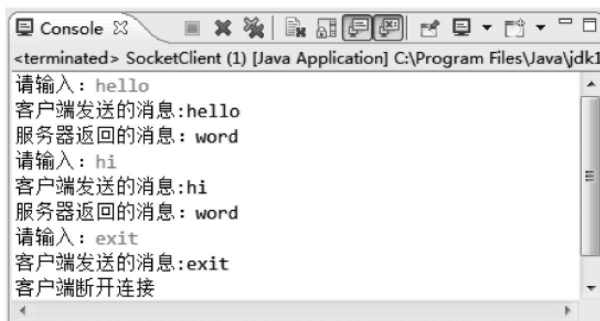


图 7-12 客户端发送退出消息

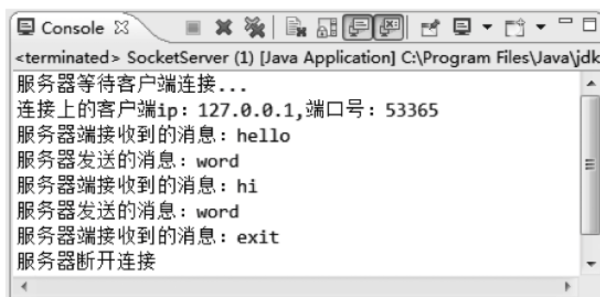


图 7-13 服务器端接收到退出消息并退出

```
package client;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;

public class SocketClient {
    public SocketClient() {
        InputStream in =null;
        OutputStream out =null;
        Socket socket = null;
        try {
            socket=new Socket("localhost",8008) ;
            out = socket.getOutputStream();
            while(true)
            {
            System.out.print("请输入:");
            Scanner scanner = new Scanner(System.in);
            String mes = scanner.next();
```

```
23                 System.out.println("客户端发送的消息:"+mes);
24                 out.write(mes.getBytes());
25                 if(mes.equals("exit"))break;
26                 in = socket.getInputStream();
27                 byte[] buffer = new byte[1024];
28                 int index = in.read(buffer);
29                 String receive = new String(buffer, 0, index);
30                 System.out.println("服务器返回的消息:"+receive);
31                 }
32                 System.out.println("客户端断开连接");
33                 in.close();
34                 out.close();
35                 socket.close();
36             } catch (UnknownHostException e) {
37                 e.printStackTrace();
38             } catch (IOException e) {
39                 e.printStackTrace();
40             }
41         }
42
43         public static void main(String[] args) {
44             new SocketClient();
45
46         }
47
48     }
```

客户端 SocketClient 在包 client 中，我们将本代码和 Chap7.6.3 工程中的 SocketClient 对比，可以发现我们将第 20 行到第 30 行，客户端与服务器进行消息发送和接收这段代码放入了一个 while 循环中，这样客户端就可以一直发送消息和获取消息，然后在第 25 行对客户端发送的消息进行判断，如果客户端发送了“exit”退出消息，则客户端跳出循环，结束程序。

下面我们来了解一下服务器端 SocketServer，代码如下所示：

```
package server;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class SocketServer {

    public SocketServer() {
```

```
11              InputStream in =null;
12              OutputStream out =null;
13              Socket socket = null;
14              try {
15                  ServerSocket serverscoket = new ServerSocket(8008);
16                  System.out.println("服务器等待客户端连接...");
17                  socket = serverscoket.accept();
18                  String ip = socket.getLocalAddress().getHostAddress();
19                  int port = socket.getPort();
20                  System.out.println("连接上的客户端 ip:" + ip + ",端口号:" + port);
21                  while(true)
22                  {
23                  in = socket.getInputStream();
24                  byte[] buffer = new byte[1024];
25                  int index = in.read(buffer);
26                  String receive = new String(buffer, 0, index);
27                  System.out.println("服务器端接收到的消息:" + receive);
28                  if(receive.equals("exit"))break;
29                  out = socket.getOutputStream();
30                  String mes = "word";
31                  out.write(mes.getBytes());
32                  System.out.println("服务器发送的消息:"+mes);
33                  }
34                  System.out.println("服务器断开连接");
35                  in.close();
36                  out.close();
37                  socket.close();
38                  serverscoket.close();
39
40              } catch (IOException e) {
41                  e.printStackTrace();
42              }
43
44          }
45
46          public static void main(String[] args) {
47              new SocketServer();
48          }
49
50      }
```

服务器端 SocketServer 在 server 包中，同样地，我们把本代码和 Chap7. 6. 3 的 SocketServer 对比，可以发现，本代码只是在 Chap7. 6. 3 的 SocketServer 收发消息这一部分增加了一个 while 循环，使服务器端可以不停地接收和响应客户端信息。在第 28 行对收到

的消息进行判断，如果收到的是客户端的退出消息，则服务器端也退出循环并且关闭连接。

本小节实现了一个客户端和一个服务器端的多次通信，那么多个客户端和一个服务器端应该如何实现呢？

7.6.5　多个客户端和一个服务器串行通信

上一小节我们讲述了一个客户端和一个服务器端的多次通信，本节将在上一小节的 Chap7.6.4 的基础上对服务器端代码进行修改，再加入一个 while 循环，使服务器端接收多个客户端。

首先运行服务端，并且运行一个客户端 Client_1，服务器端如图 7-14 所示。

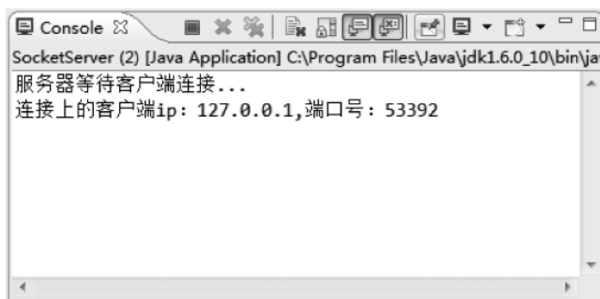


图 7-14　服务器端连入一个客户端

然后客户端 Client_1 发送一条消息给服务器端，服务器如图 7-15 所示。

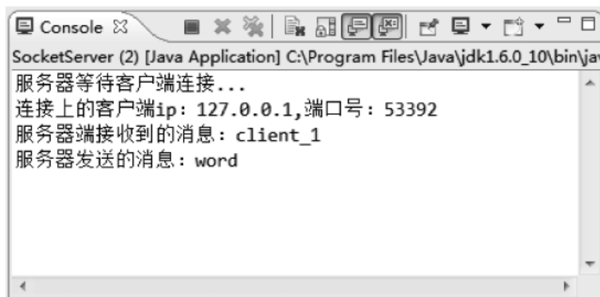


图 7-15　服务器端

此时我们再次运行一个客户端 Client_2，此时服务器端如图 7-16 所示。

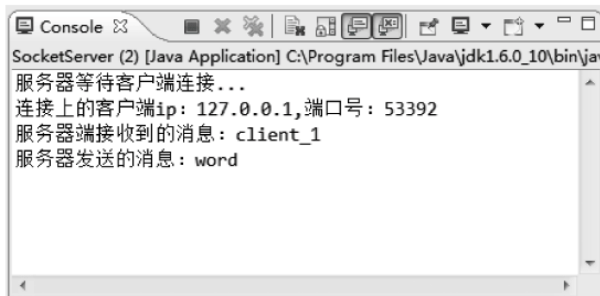


图 7-16　启动 Client_2 后

我们发现客户端 Client_2 没有和服务器端建立连接，为了进一步验证，用客户端 Client_2 向服务器端发送一条消息，客户端 Client_2 如图 7-17 所示。

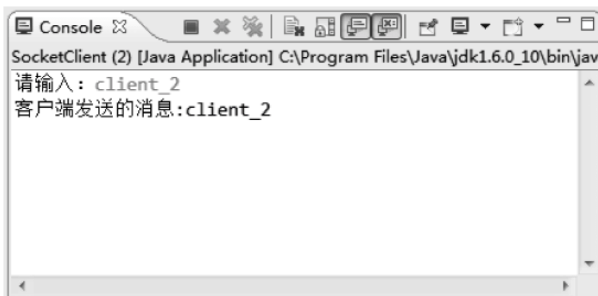


图 7-17　客户端 Client_2(一)

我们可以看到客户端 Client_2 发送消息后并没有收到服务器端的响应信息，证明了客户端 Client_2 确实没有和服务器端建立连接。下面我们分析客户端 Client_1 断开与服务器端的连接，如图 7-18 所示。

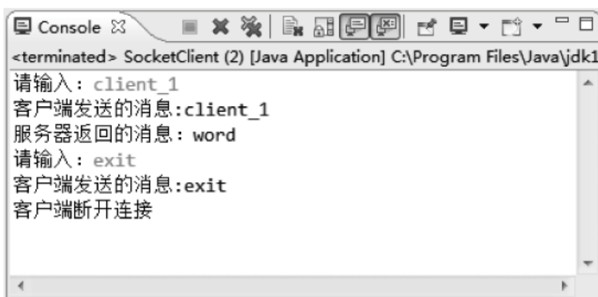


图 7-18　Client_1 断开连接

此时服务器端和客户端 Client_2 发生了变化，服务器端如图 7-19 所示，客户端 Client_2 如图 7-20 所示。

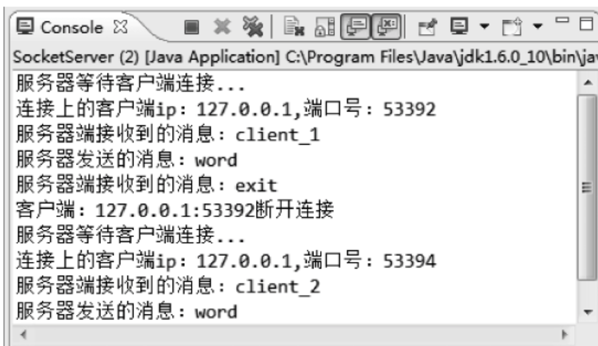


图 7-19　服务器端

我们可以看到，当客户端 Client_1 断开与服务器端的连接后，服务器端立马与客户端 Client_2 建立连接，收到客户端 Client_2 的消息并返回响应消息。

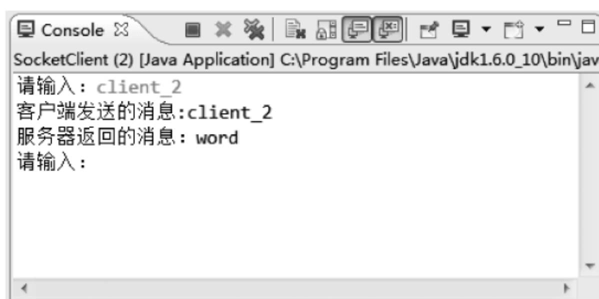


图 7-20　客户端 Client_2(二)

服务器端 SocketServer 代码如下所示：

```
package server;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class SocketServer {

    public SocketServer() {
        InputStream in =null;
        OutputStream out =null;
        Socket socket = null;
        try {
            ServerSocket serverscoket = new ServerSocket(8008);
        while(true)
        {
            System.out.println("服务器等待客户端连接...");
            socket = serverscoket.accept();
            String ip = socket.getLocalAddress().getHostAddress();
            int port = socket.getPort();
            System.out.println("连接上的客户端 ip:" + ip + ",端口号:" + port);
            while(true)
            {
            in = socket.getInputStream();
            byte[] buffer = new byte[1024];
            int index = in.read(buffer);
            String receive = new String(buffer, 0, index);
            System.out.println("服务器端接收到的消息:" + receive);
            if(receive.equals("exit"))break;
            out = socket.getOutputStream();
            String mes = "word";
```

```
33                  out.write(mes.getBytes());
34                  System.out.println("服务器发送的消息:"+mes);
35                  }
36                  System.out.println("客户端:"+ip+":"+port+"断开连接");
37                  in.close();
38                  out.close();
39              }
40
41              } catch (IOException e) {
42                  e.printStackTrace();
43              }
44
45          }
46
47          public static void main(String[] args) {
48              new SocketServer();
49          }
50
51  }
```

服务器端 SocketServer 在 server 包中,我们在 Chap7.6.4 的基础上在第 16 行新增加了一个 while 循环,将服务器端等待客户端的连接这段代码放入 while 循环中。可是我们通过以上事例可以看出,服务器端是没有办法在与客户端 Client_1 通信的时候同时等待其他客户端的连接和通信,只有当客户端 Client_1 退出后,服务器端才能与客户端 Client_2 进行通信。

那么怎么才能实现服务器端一边等待其他客户端的通信,一边和已连上的客户端进行通信呢? 在下一小节,我们将实现此功能。

7.6.6 多个客户端和一个服务器并行通信

上一小节我们想要实现多个客户端和一个服务器多次通信,可是我们却遇到了一个问题,服务器端不能在等待其他客户端的连入的同时和已连入的客户端通信,在本节我们通过 Java 多线程解决这个问题。

首先我们运行 Chap7.6.6 的服务器端 SocketServer 和客户端 Client_1,如图 7-21 所示。

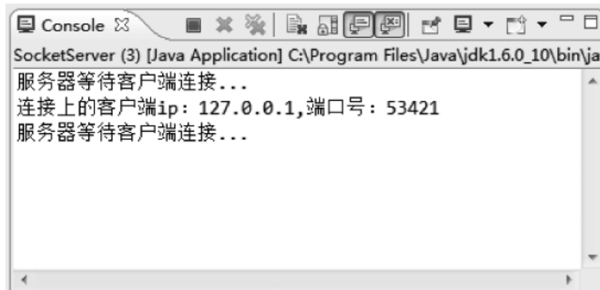


图 7-21 接入一个客户端

我们再次打开一个客户端 Client_2，此时观察服务器端的控制台，如图 7-22 所示。

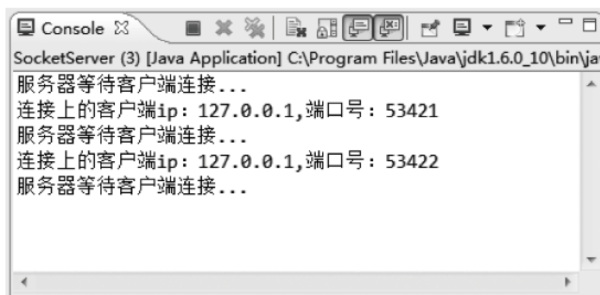


图 7-22　接入两个客户端

和上一小节的 Chap7.6.5 运行结果相比较，我们可以发现本工程的服务器端可以接收两个客户端同时连接，也可以与客户端 Client_1 和客户端 Client_2 进行通信，而不需要关闭其中一个客户端，如图 7-23 所示。

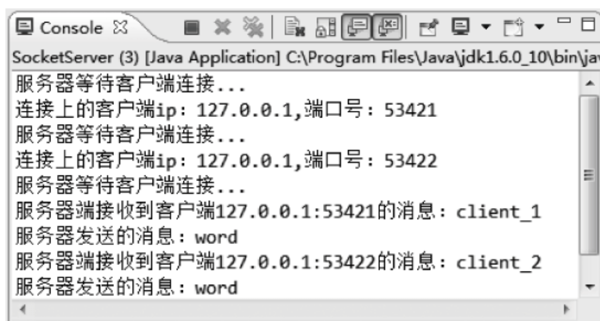


图 7-23　与不同客户端通信

下面我们来了解一下是如何实现本工程的。本工程是在 Chap7.6.4 的基础上对服务器端进行了修改，当每次有客户端连入时，都会在服务器端为这个客户端开一个独立的线程收发消息，服务器端 SocketServer 代码如下所示：

```
package server;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class SocketServer {

    public SocketServer() {
        Socket socket = null;
        try {
            ServerSocket serverscoket = new ServerSocket(8008);
            while(true)
            {
                System.out.println("服务器等待客户端连接...");
```

```
15                  socket = serverscoket.accept();
16                  String ip = socket.getLocalAddress().getHostAddress();
17                  int port = socket.getPort();
18                  System.out.println("连接上的客户端 ip:" + ip + ",端口号:" + port);
19                  new ServerThread(socket).start();
20              }
21
22          } catch (IOException e) {
23              e.printStackTrace();
24          }
25
26      }
27
28      public static void main(String[] args) {
29          new SocketServer();
30      }
31  }
```

与 Chape5.2.4 的 SocketServer 对比，我们在第 19 行新增加了一个开启线程的功能，服务器端通过第 12 行的 while 循环不断接收客户端的请求连入，当客户端连入后，在第 19 行为连入的客户端开启一个新的线程用于与客户端进行通信，此时服务器端又回到第 15 行继续等待下一个客户端的请求连入。我们就是通过开启新线程的方式解决了工程 Chap7.6.5 服务器端不能同时接收多个客户端的连入，线程 ServerThread 的代码如下所示：

```
package server;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

public class ServerThread extends Thread {
        private Socket socket;

    public ServerThread(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        InputStream in = null;
        OutputStream out = null;
        String ip = socket.getLocalAddress().getHostAddress();
        int port = socket.getPort();
        try {
```

```
                while (true) {
                    in = socket.getInputStream();
                    byte[] buffer = new byte[1024];
                    int index = in.read(buffer);
                    String receive = new String(buffer, 0, index);
                    System.out.println("服务器端接收到客户端
                    "+ip+":"+port+"的消息:" + receive);
                    if (receive.equals("exit"))
                        break;
                    out = socket.getOutputStream();
                    String mes = "word";
                    out.write(mes.getBytes());
                    System.out.println("服务器发送的消息:" + mes);
                }
                System.out.println("客户端:"+ip+":"+port+"断开连接");
                in.close();
                out.close();
                socket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }

        }
    }
```

ServerThread 在 server 包中，ServerThread 是一个线程类，因为它继承了 Thread(注意：实现了 Runnable 接口的类也是线程类)，并且覆写了 run 方法，在 run 方法中实现了与客户端之间的通信，实现通信的方法和 Chap7.6.4 是一样的，这里就不作过多阐述了。

7.6.7 客户端与服务器端 HTTP 通信

上几节我们分析了客户端与服务器端的 TCP 通信方式，通过 Socket 连接，本节我们讲述 Http 通信，Chap7.6.7 使用 tomcat 作为服务器。下面运行 Chap7.6.7 的服务器端 HttpServer 和客户端的 HttpClient，并且在客户端输入“hello”字符串，客户端如图 7-24 所示，服务器端如图 7-25 所示。

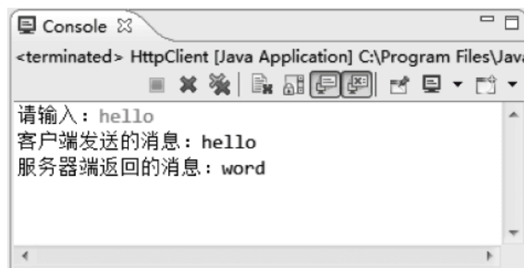


图 7-24 Http 客户端



图 7-25　Http 服务器端

Http 和 TCP 的区别是：HTTP 在每次请求结束后都会主动释放连接，因此 HTTP 连接是一种"短连接"，要保持客户端程序的在线状态，需要不断地向服务器发起连接请求。而 TCP 连接是一种"长连接"，它们之间的连接并不会主动关闭，后续的读写操作会继续使用这个连接。下面我们来讲解 Chap7.6.7 中的 HttpClient 和 HttpServer 的实现过程，首先是 HttpClient，代码如下所示：

```
package client;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Scanner;



public class HttpClient{

    public HttpClient() {
        System.out.print("请输入:");
        Scanner scanner = new Scanner(System.in);
        String mes = scanner.next();
        String urlStr = "http://localhost/Chap7.6.7/HttpServer";
        URL url;
        try {
            url = new URL(urlStr);
            HttpURLConnection connection = (HttpURLConnection) url
                    .openConnection();
            connection.setDoOutput(true);
            connection.setRequestMethod("POST");
            OutputStream out = connection.getOutputStream();
            System.out.println("客户端发送的消息:"+mes);
            out.write(mes.getBytes());
            InputStream in = connection.getInputStream();
```

```
29                  byte[] buffer = new byte[1024];
30                  int index = in.read(buffer);
31                  String receive = new String(buffer, 0, index);
32                  System.out.println("服务器端返回的消息:"+receive);
33                  in.close();
34                  out.close();
35              } catch (IOException e) {
36                  e.printStackTrace();
37              }
38
39      }
40
41      public static void main(String[] args) {
42                      new HttpClient();
43
44                      }
45
46  }
```

HttpClient 在 client 包中，第 17 行定义了 URL 地址，在第 20 行定义了 url 对象，在第 21 行得到 HttpURLConnection 对象，第 23 行设定。将 doOutput 标志设置为 true，指示应用程序要将数据写入 URL 连接，在第 24 行设定请求的方式是 POST 请求，在第 25 行和第 28 行得到输出流和输入流，然后与服务器端交互。

接下来是 HttpServer，代码如下所示：

```
package server;

import java.io.IOException;
import javax.servlet.ServletInputStream;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class HttpServer extends HttpServlet{

    public void doPost(HttpServletRequest req, HttpServletResponse resp)
    {
            try {
              ServletInputStream in = req.getInputStream();
              ServletOutputStream out = resp.getOutputStream();
                int len = req.getContentLength();
                byte[] buffer = new byte[len];
```

```
20                 int index = in.read(buffer);
21                 String receive = new String(buffer, 0, index);
22                System.out.println("服务器端接收到的消息:"+receive);
23                String mes="word";
24                out.write(mes.getBytes());
25                System.out.println("服务器端发送的消息:"+mes);
26                 in.close();
27                 out.close();
28            } catch (IOException e) {
29                e.printStackTrace();
30            }
31
32        }
33  }
```

HttpServer 在 server 包中,HttpServer 继承了 HttpServlet,并且覆写了 doPost()方法,第 16 行和第 17 行得到了输入输出流,和客户端进行交互。客户端发送 post 请求,tomcat 再将 HttpServer 放入 tomcat 中运行,并且根据请求的方式执行请求的 post 方法,然后断开连接。

7.7 图书管理系统 V7.0

我们只给出了增加图书和删除图书的部分,剩下部分请同学们自行完成。

7.7.1 运行效果图

图 7-26、图 7-27 和图 7-28 展示了图书管理系统 V7.0 的部分运行效果。



图 7-26 客户端与服务器分离的图书管理系统客户端运行截图(一)

7.7.2 类结构示意图

图 7-29 和图 7-30 分别展示了客户端和服务器端的类结构,图 7-31 和图 7-32 分别展示了客户端和服务器端的类设计结构。



图 7-27　客户端与服务器分离的图书管理系统客户端运行截图(二)

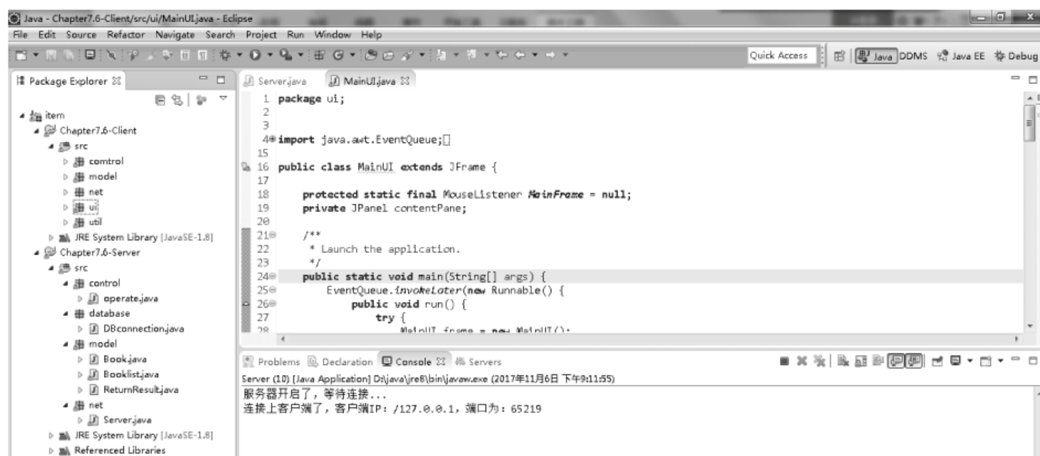


图 7-28　客户端与服务器分离的图书管理系统服务器运行截图(三)

1. 类结构

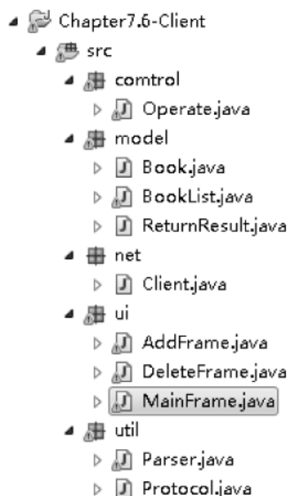


图 7-29　客户端类结构图

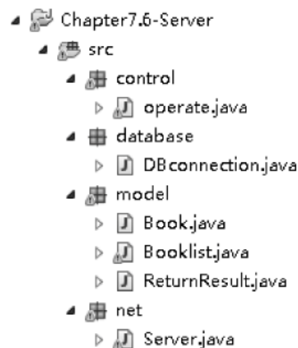


图 7-30　服务器端类结构图

2. 客户端/服务器端类设计结构

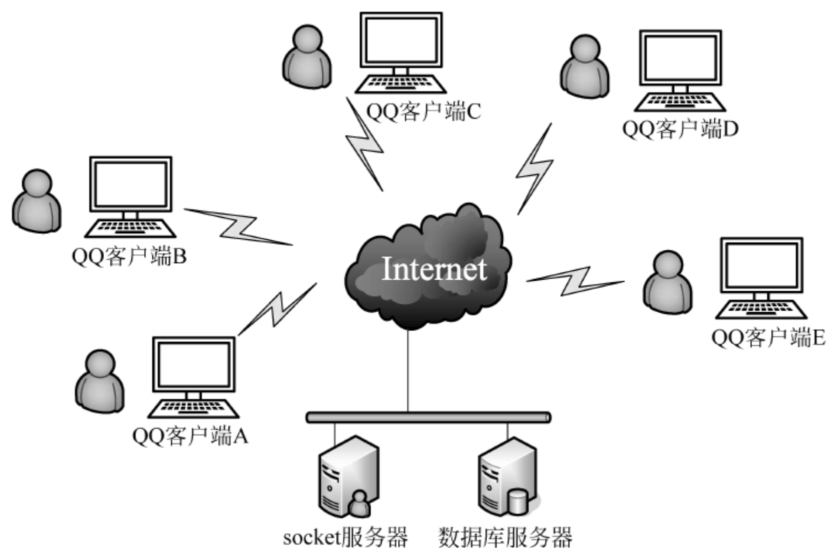


图 7-31　客户端类设计结构图

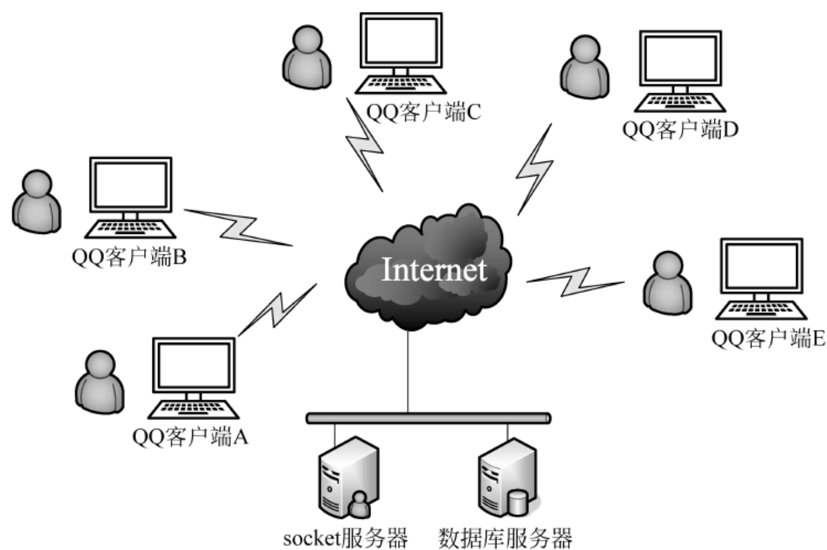


图 7-32　服务器端类设计结构图

7.7.3　通信协议

客户端服务端需要与服务器端进行通信，那么首先需要自定义客户端和服务器端的通信协议，协议如下。

1. 增加图书

客户端：

operate：add

content：图书

服务器：

result：Success/notSuccess

2. 删除图书

客户端：

operate：delete

content：图书名

服务器：

result：Success/notSuccess

7.7.4 关键代码

1. 客户端代码

Book.java

```
package model;

public class Book {

    private String bookname;

    private String author;

    private float price;

    public Book(String bookname, String author, float price)
    {
        this.bookname = bookname;
        this.author = author;
        this.price = price;
    }

    public String getBookname() {
        return bookname;
    }

    public String getAuthor() {
        return author;
    }

    public float getPrice() {
        return price;
    }
}
```

BookList.java

```
package model;

import java.util.Vector;

public class BookList extends Vector<Book> {

}
```

ReturnResult.java

```
package model;

public class ReturnResult {

    private boolean isSuccess;

    private String failreason;

    private Object returnData;

    public boolean isSuccess() {
        return isSuccess;
    }

    public void setSuccess(boolean isSuccess) {
        this.isSuccess = isSuccess;
    }

    public String getFailreason() {
        return failreason;
    }

    public void setFailreason(String failreason) {
        this.failreason = failreason;
    }

    public Object getReturnData() {
        return returnData;
    }

    public void setReturnData(Object returnData) {
        this.returnData = returnData;
    }
}
```

MainFrame.java

```
package ui;
```

```

import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import net.Client;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.MouseListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.ActionEvent;

public class MainFrame extends JFrame {

    protected static final MouseListener MainFrame = null;
    private JPanel contentPane;

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    MainFrame frame = new MainFrame();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    public MainFrame() {
        setTitle("图书管理系统");

        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                Client.sendMessageToServer("closeserver");
                System.exit(0);
            }
        });
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
```

```
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JButton btnNewButton = new JButton("增加图书");
        btnNewButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                AddFrame addbook = new AddFrame();
                addbook.setVisible(true);
                MainFrame.this.dispose();
            }
        });
        btnNewButton.setBounds(47, 38, 93, 23);
        contentPane.add(btnNewButton);

        JButton btnNewButton_1 = new JButton("删除图书");
        btnNewButton_1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                DeleteFrame deletebook = new DeleteFrame();
                deletebook.setVisible(true);
                MainFrame.this.dispose();

            }
        });
        btnNewButton_1.setBounds(199, 38, 93, 23);
        contentPane.add(btnNewButton_1);

        JButton btnNewButton_2 = new JButton("修改图书");
        btnNewButton_2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

            }
        });
        btnNewButton_2.setBounds(47, 111, 93, 23);
        contentPane.add(btnNewButton_2);

        JButton btnNewButton_3 = new JButton("查找图书");
        btnNewButton_3.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

            }
        });
        btnNewButton_3.setBounds(199, 111, 93, 23);
        contentPane.add(btnNewButton_3);

```

```
        JButton btnNewButton_4 = new JButton("退出");
        btnNewButton_4.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                MainFrame.this.dispose();
                Client.sendMessageToServer("closeserver");

            }
        });
        btnNewButton_4.setBounds(122, 189, 93, 23);
        contentPane.add(btnNewButton_4);
    }
}
```

AddFrame.java

```
package ui;

import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import comtrol.Operate;
import model.Book;
import model.ReturnResult;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.ActionEvent;

public class AddFrame extends JFrame {

    private JPanel contentPane;
    private JTextField AddnametextField;
    private JTextField AddauthortextField;
    private JTextField AddpricetextField;

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    AddFrame frame = new AddFrame();
                    frame.setVisible(true);
```

```
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
    public AddFrame() {
        setTitle("增加图书界面");
        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                MainFrame frame = new MainFrame();
                frame.setVisible(true);
                AddFrame.this.dispose();
            }
        });
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JLabel label = new JLabel("图书名称：");
        label.setBounds(60, 39, 95, 15);
        contentPane.add(label);

        AddnametextField = new JTextField();
        AddnametextField.setBounds(214, 36, 66, 21);
        contentPane.add(AddnametextField);
        AddnametextField.setColumns(10);

        JLabel lblNewLabel = new JLabel("作者名称：");
        lblNewLabel.setBounds(60, 96, 95, 15);
        contentPane.add(lblNewLabel);

        AddauthortextField = new JTextField();
        AddauthortextField.setBounds(214, 93, 66, 21);
        contentPane.add(AddauthortextField);
        AddauthortextField.setColumns(10);

        JLabel lblNewLabel_1 = new JLabel("图书价格：");
        lblNewLabel_1.setBounds(60, 153, 95, 15);
        contentPane.add(lblNewLabel_1);

```

```
        AddpricetextField = new JTextField();
        AddpricetextField.setBounds(214, 150, 66, 21);
        contentPane.add(AddpricetextField);
        AddpricetextField.setColumns(10);

        JButton btnNewButton = new JButton("确定");
        btnNewButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String bookname = AddnametextField.getText();
                String author = AddauthortextField.getText();
                String Oprice = AddpricetextField.getText();
                float price = Float.parseFloat(Oprice);
                Book book = new Book(bookname, author, price);
                Operate operator = new Operate();
                ReturnResult isSuccess = operator.addBook(book);
                if (isSuccess.isSuccess()) {
                    JOptionPane.showMessageDialog(null, "添加图书成功!");

                } else {
                    JOptionPane.showMessageDialog(null, "添加图书失败!");
                }

            }
        });
        btnNewButton.setBounds(70, 206, 93, 23);
        contentPane.add(btnNewButton);

        JButton btnNewButton_1 = new JButton("退出");
        btnNewButton_1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                MainFrame frame = new MainFrame();
                frame.setVisible(true);
                AddFrame.this.dispose();

            }
        });
        btnNewButton_1.setBounds(197, 206, 93, 23);
        contentPane.add(btnNewButton_1);
    }
}
```

DeleteFrame.java

```
package ui;

import java.awt.BorderLayout;
```

```
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import control.Operate;
import model.ReturnResult;
import net.Client;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.ActionEvent;

public class DeleteFrame extends JFrame {

    private JPanel contentPane;
    private JTextField DeletetextField;

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    DeleteFrame frame = new DeleteFrame();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    public DeleteFrame() {
        setTitle("删除图书界面");
        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                MainFrame frame = new MainFrame();
                frame.setVisible(true);
                DeleteFrame.this.dispose();
            }
        });
```

```
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JLabel lblNewLabel = new JLabel("删除书名：");
        lblNewLabel.setBounds(73, 71, 108, 15);
        contentPane.add(lblNewLabel);

        DeletetextField = new JTextField();
        DeletetextField.setBounds(230, 68, 66, 21);
        contentPane.add(DeletetextField);
        DeletetextField.setColumns(10);

        JButton btnNewButton = new JButton("确定");
        btnNewButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String bookname = DeletetextField.getText();
                Operate operator = new Operate();
                ReturnResult isSuccess = operator.deletebook(bookname);
                if (isSuccess.isSuccess()) {
                    JOptionPane.showMessageDialog(null, "删除图书成功!");
                } else {
                    JOptionPane.showMessageDialog(null, "删除图书失败!");
                }
            }
        });
        btnNewButton.setBounds(88, 153, 93, 23);
        contentPane.add(btnNewButton);

        JButton btnNewButton_1 = new JButton("退出");
        btnNewButton_1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                MainFrame frame = new MainFrame();
                frame.setVisible(true);
                DeleteFrame.this.dispose();

            }
        });
        btnNewButton_1.setBounds(224, 153, 93, 23);
        contentPane.add(btnNewButton_1);
    }
}
```



Operate.java

```
package comtrol;

import model.Book;
import model.ReturnResult;
import net.Client;
import util.Parser;
import util.Protocol;

public class Operate {

    public ReturnResult addBook(Book book) {
        Protocol protocol = new Protocol();
        String message = protocol.addrMessage(book);
        String returnMessage = Client.sendMessageToServer(message);
        Parser parser = new Parser();
        ReturnResult returnResult = parser.parseraddbookresult(returnMessage);
        return returnResult;
    }

    public ReturnResult deletebook(String bookname) {
        Protocol protocol = new Protocol();
        String message = protocol.deleteMessage(bookname);
        String returnMessage = Client.sendMessageToServer(message);
        Parser parser = new Parser();
        ReturnResult returnResult = parser.parserdeletebookresult(returnMessage);
        return returnResult;
    }

}
```

Protocol.java

```
package util;

import model.Book;

public class Protocol {

    public String addrMessage(Book book) {
        String addbook = "operate:add/n" + book.getBookname() + "," + book.
        getAuthor() +  "," + book.getPrice();
        System.out.println(addbook);
        return addbook;
    }
```

```
    public String deleteMessage(String bookname) {
        String delete = "operate:delete/n" + bookname;
        return delete;
    }
}
```

Client.java

```
package net;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
import java.net.UnknownHostException;

public class Client {

    static byte[] buffer = new byte[15000];
    static Socket tcpConnection;
    static InputStream in;
    static OutputStream out;
    static {
        try {
            tcpConnection = new Socket("127.0.0.1", 1234);
            System.out.println("接上服务器了");
            in = tcpConnection.getInputStream();
            out = tcpConnection.getOutputStream();
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static String sendMessageToServer(String message) {
        String str = null;
        try {
            System.out.println("向服务器发送了：" + message);
            out.write(message.getBytes());
            int count = in.read(buffer);
            str = new String(buffer, 0, count);
            System.out.println("从服务器返回了：" + str);
            if (str.equals("close")) {
                closeClient();
```

```
                }
                return str;
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
            }
            return str;
        }

        public static void closeClient() //关闭客户端并关闭连接
        {
            try {
                in.close();
                out.close();
                tcpConnection.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
}
```

Parser.java

```
package util;

import model.ReturnResult;

public class Parser {
    public ReturnResult parseraddbookresult(String returnMessage) {
        ReturnResult returnResult = new ReturnResult();
        if (returnMessage.equals("result:Success")) {
            returnResult.setSuccess(true);
            return returnResult;
        } else {
            returnResult.setSuccess(false);
            // returnResult.failreason =...;
            return returnResult;
        }
    }

    public ReturnResult parserdeletebookresult(String message) {
        ReturnResult returnResult = new ReturnResult();
        System.out.println(message);
        if (message.equals("result:Success")) {
            returnResult.setSuccess(true);
            return returnResult;
```

```java
            } else {
                returnResult.setSuccess(false);
                // returnResult.failreason =...;
                return returnResult;
            }

        }
}
```

2. 服务器端代码

服务器端 Model 层的 Book. java、BookList. java、ReturnResult. java 与客户端一致。

Operate.java

```java
package control;

import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import database.DBconnection;
import model.Book;
import model.ReturnResult;

public class operate {

    public ReturnResult addBook(Book book) {
        ReturnResult result = new ReturnResult();
        int line = 0;
        try {
            Connection conn = DBconnection.getConnection();
            Statement s;
            s = conn.createStatement();
            String sql = "insert into booklist (bookname,author,price) values('" +
            book.bookname + "','" + book.author+ "'," + book.price + ")";

            line = s.executeUpdate(sql);

        } catch (SQLException e) {
            e.printStackTrace();
        }
        if (line > 0) {
            result.setSuccess(true);
            return result;
        } else {

            result.setFailreason("□∞");
```




```
                return result;
            }
        }

        public ReturnResult deletebook(String bookname) {
            ReturnResult result = new ReturnResult();
            int line = 0;
            Connection conn = DBconnection.getConnection();
            try {
                Statement s = conn.createStatement();
                String sql = "delete from booklist where bookname = '" + bookname + "'";
                line = s.executeUpdate(sql);
            } catch (SQLException e) {
                e.printStackTrace();
            }
            if (line > 0) {
                result.setSuccess(true);
                return result;
            } else {

                result.setFailreason("□∞");
                return result;
            }

        }
    }
```

DBconnection .java

```
package database;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DBconnection {

    static final String url =
"jdbc:mysql://localhost:3306/book?characterEncoding=utf8&useSSL=true";
    static final String name = "com.mysql.jdbc.Driver";
    static final String user = "root";
    static final String password = "mysql123";

    static {
```

```
        try {
            Class.forName(name);                //指定连接类型
        } catch (ClassNotFoundException cE) {
            System.out.println("Class Not Found Exception:" + cE.toString());
        }
    }

    public static Connection getConnection() {
        try {
            return DriverManager.getConnection(url, user, password);
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }

    public static void closeConnection(ResultSet rs, Statement statement, Connection con) {
        try {
            if (rs != null)
                rs.close();
            if (statement != null)
                statement.close();
            if (con != null)
                con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public static void closeConnection(Statement statement, Connection con) {
        closeConnection(null, statement, con);
    }

    public static void main(String[] args) {

    }
}
```

Server.java

```
package net;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
```

```
import java.net.Socket;

import control.operate;
import model.Book;
import model.Booklist;
import model.ReturnResult;

public class Server {

    byte[] buffer = new byte[256];

    public Server() {
        ServerSocket ss = null;
        Socket tcpConnection = null;
        InputStream in = null;
        OutputStream out = null;

        try {
            ss = new ServerSocket(1234);
            System.out.println("服务器开启了,等待连接...");
            tcpConnection = ss.accept();
            System.out.println("连接上客户端了,客户端 IP: " + tcpConnection.
            getInetAddress() + ",端口为: " + tcpConnection.getPort());
            in = tcpConnection.getInputStream();
            out = tcpConnection.getOutputStream();
            while (true) {
                Booklist booklist = new Booklist();
                int count = in.read(buffer);
                String str = new String(buffer, 0, count);
                if (str.equals("closeserver")) {
                    break;
                }
                String[] totalstr = str.split("/n");
                String head = totalstr[0];
                String tail = totalstr[1];
                if (head.equals("add")) {
                    operate operator = new operate();
                    String[] strArray = null;
                    strArray = tail.split(",");
                    String bookname = strArray[0];
                    String author = strArray[1];
                    String aprice = strArray[2];
                    float price = Float.parseFloat(aprice);
                    Book book = new Book(bookname, author, price);
```

```
                    ReturnResult returnresult = new ReturnResult();
                    returnresult = operator.addBook(book);
                    if (returnresult.isSuccess()) {
                        out.write("Success".getBytes());
                    } else {
                        out.write("notSuccess".getBytes());
                    }
                }
                if (head.equals("delete")) {
                    operate operator = new operate();
                    ReturnResult returnresult = new ReturnResult();
                    returnresult = operator.deletebook(tail);
                    if (returnresult.isSuccess()) {
                        out.write("Success".getBytes());
                    } else {
                        out.write("notSuccess".getBytes());
                    }
                }
            }
            out.write("close".getBytes());
            in.close();
            out.close();
            tcpConnection.close();

        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {
        new Server();
    }
}
```

第 8 章

多 线 程

计算机能够拥有强大的功能，在很大程度上，都依赖于即将在本章讲述的多线程概念。让我们来了解一下多线程是怎么让 Java 有“三头六臂”，完成分配给它的多个任务的。

8.1 本 章 任 务

理论任务：掌握线程的本质、生命周期和实现方式。

实践任务：把上一章的服务器并行化，让多个客户端能同时连接服务器。

8.2 几 个 概 念

8.2.1 进程

1. 进程概述

大家使用计算机的习惯大多是：打开计算机，一边写程序或者逛网页，一边听音乐或者在 QQ 上与好友闲聊。无论是浏览器、编程软件，还是 QQ，它们都是独立的系统进程。正是因为计算机在多年进化的历程中，拥有了多任务这一技术，所以才能够实现同时操作多个软件。想象一下，如果没有这个技术，使用计算机时将会有多么乏味。在 Windows 中，每一个打开运行的应用程序或后台程序都是一个进程。我们感觉这些程序是“同时”运行的，但实际上，一个处理器同一时刻只能运行一个进程，只是 CPU 在高速轮换执行让我们有这样的错觉，我们感受不到中断的原因是 CPU 执行速度相对于我们的感觉实在是太快了。

在本章节，我们将会探究计算机在进行多任务操作时，程序是以何种方式存在于系统中的。首先，我们需要打开任务管理器，在管理器中，每个正在运行的程序都会一一罗列。从中我们便可看出每个程序都是以一个独立的进程作为依托，通过进程，程序可以使用系统资源，有操作系统进行调度，从而服务于用户。

那么，到底什么是进程呢？进程是出现在操作系统中基础而又极为重要的一个概念。它是正在运行的程序的实例。每一个进程都有它自己的地址空间，一般情况下，包括文本区域(text region)、数据区域(data region)和堆栈(stack region)。文本区域存储处理器执行的代码；数据区域存储变量和进程执行期间使用的动态分配的内存；堆栈区域存储着活动过程调用的指令和本地变量。

2. 进程特点

(1) 独立性：每个进程拥有自己独立的资源，拥有私有的地址空间，其大小与处理机位数有关，如 Win32 系统，32 位地址映射 4GB 地址空间，其中低地址的 2GB 作为用户模式的虚拟地址空间(应用程序可共享，线程间独立)，高地址的 2GB 作为内核模式的虚拟地址空间(系统使用)。

(2) 动态性：这点从进程的概念可以看出，运行中的程序就是进程。进程中有时间、状态(博文详解)、生命周期等动态的概念。

(3) 并发性：多个进程在单个处理器上并发执行。

3. 进程内容

(1) 程序代码：用于描述进程要完成的功能。

(2) 数据集合：程序执行所需要的数据与工作区域。

(3) PCB 程序控制块：包含进程的描述信息与控制信息，是进程的唯一标志，也正是因为有了 PCB，进程就成了一个动态的概念。

8.2.2 线程

1. 线程概述

我们了解了进程的概念，知道了它是一个程序的示例。那么，我们也就明白了，在计算机世界里，每个程序都是一个个公民，其生来平等，享有计算机各项资源的权利。但它们使用的资源也是有条件的——必须为用户或者操作系统服务。所以，它们是公民也是一个个工人，它们必须在自己的工作岗位上各司其职。工作繁重的时候，处理一些小问题时并不需要进程自己动手，它只需要召唤自己的小弟，代替它完成这些工作即可。

既然小弟能够代理大哥很好地解决小问题，那么它也算是劳模同志。我们也就来介绍一下站在大哥身后的小弟——线程。一个标准的线程由线程 ID、当前指令指针(PC)、寄存器集合和堆栈组成。另外，线程是进程中的一个实体，是被系统独立调度和分派的基本单位，线程自己不拥有系统资源，只拥有一点儿在运行中必不可少的资源，但它可与同属一个进程的其他线程共享进程拥有的全部资源。

我们都有过这样的经历：当我们去食堂排队打饭时，如果只有一个窗口，那么所有的人都要去那一个窗口打饭。在同等时间的情况下，如果开多个窗口，那么每个窗口都可以打饭。这里的一个窗口就好比一个单线程，多个窗口就类似开了多个线程，而每一个打饭的师傅就好比 CPU，我们就好比一个又一个的应用程序，所以线程的本质其实就是提高 CPU 利用率。

2. 线程的特点

(1) 轻型实体(线程的实体包括程序、数据和 TCB。TCB 是用于指示被执行指令序列的程序计数器、保留局部变量、少数状态参数和返回地址等的一组寄存器和堆栈)。

(2) 独立调度和分派的基本单位。

(3) 可并发执行。

(4) 共享进程资源(所有线程都具有相同的地址空间(进程的地址空间)，这意味着，线程可以访问该地址空间的每一个虚地址；此外，还可以访问进程所拥有的已打开文件、

定时器、信号量机构等。由于同一个进程内的线程共享内存和文件，所以线程之间互相通信不必调用内核）。

8.3 生命周期

8.3.1 线程生命周期概述

与人有生老病死一样，线程同样也要经历不同的生命阶段。当线程被创建并启动以后，它既不是一启动就进入执行状态，也不是一直处于执行状态。在线程的生命周期中，它要经过新建(New)、就绪(Runnable)、运行(Running)、阻塞(Blocked)和死亡(Dead)五种状态。尤其是当线程启动以后，它不可能一直“霸占”着 CPU 独自运行，所以 CPU 需要在多条线程之间切换，于是线程状态也会多次在运行、阻塞之间切换。以下是线程的几种状态。

(1) 新建状态：当程序使用 new 关键字创建了一个线程之后，该线程就处于新建状态，此时仅由 JVM 为其分配内存，并初始化其成员变量的值。

(2) 就绪状态：当线程对象调用了 start()方法之后，该线程处于就绪状态。Java 虚拟机会为其创建方法调用栈和程序计数器，等待调度运行。

(3) 运行状态：如果处于就绪状态的线程获得了 CPU，开始执行 run()方法的线程执行体，则该线程处于运行状态。

(4) 阻塞状态：当处于运行状态的线程失去所占用资源之后，便进入阻塞状态。

(5) 死亡状态：线程执行完了或者因异常退出了 run()方法，该线程结束生命周期。

8.3.2 为什么要有生命周期

为什么需要生命周期呢？其实自然界中的各种物质都是有生命周期的，人也一样，从生到死。当然，生命周期不只生、死两个状态。就像我们人一样，我们一辈子有很多状态。之所以需要生命周期，是方便操作系统管理线程。

先作个比喻，上幼儿园的小朋友，如果犯了错误，不听老师话，那么老师会打电话给他爸爸；如果生病了，那么老师会打电话给他妈妈；如果遇到了坏人就打 110，等等。这只是个比喻，方便读者理解，可能不恰当或不适合现代幼儿教学理念。这里老师就是操作系统，负责在孩子的不同状态下通知调用不同的人(接口)。孩子就是各个不同的线程，爸爸、妈妈、110 就是系统定义，由系统调用，供程序员重写的接口。至于爸爸做什么、妈妈做什么、110 来了做什么，那是应用程序员的事，如图 8-1 所示。

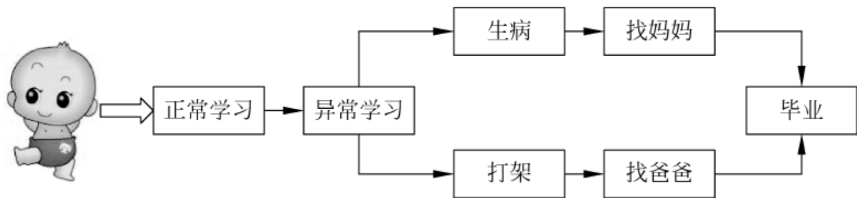


图 8-1 孩子入学幼儿园举例

学过 Java Web 的同学会想到，servlet 也有生命周期，所以说 Tomcat 是 servlet 容器。这和线程是类似的。

8.3.3 线程生命周期详解

1. 新建和就绪状态

当程序使用 new 关键字创建了一个线程之后，该线程就处于新建状态，此时它和其他的 Java 对象一样，仅仅由 Java 虚拟机为其分配内存，并初始化其成员变量的值。此时的线程对象没有表现出任何线程的动态特征，程序也不会执行线程的线程执行体。

当线程对象调用了 start()方法之后，该线程处于就绪状态。Java 虚拟机会为其创建方法调用栈和程序计数器，处于这个状态中的线程并没有开始运行，只是表示该线程可以运行了。至于该线程何时开始运行，取决于 JVM 里线程调度器的调度。

注意：启动线程使用 start()方法，而不是 run()方法。永远不要调用线程对象的 run()方法。调用 start0 方法来启动线程，系统会把该 run()方法当成线程执行体来处理；但如果直接调用线程对象的 run()方法，则 run()方法立即就会被执行，而且在 run()方法返回之前其他线程无法并发执行。也就是说，系统把线程对象当成一个普通对象，而 run()方法也是一个普通方法，而不是线程执行体。需要指出的是，调用了线程的 run()方法之后，该线程已经不再处于新建状态，不要再次调用线程对象的 start()方法。只能对处于新建状态的线程调用 start()方法，否则将引发 IllegaIThreadStateExccption 异常。

调用线程对象的 start()方法之后，该线程立即进入就绪状态——就绪状态相当于“等待执行”，但该线程并未真正进入运行状态。如果希望调用子线程的 start()方法后子线程立即开始执行，程序可以使用 Thread. sleep(1)来让当前运行的线程(主线程)睡眠 1 毫秒，1 毫秒就够了，因为在这 1 毫秒内 CPU 不会空闲，它会去执行另一个处于就绪状态的线程，这样就可以让子线程立即开始执行。

2. 运行和阻塞状态

如果处于就绪状态的线程获得了 CPU，开始执行 run()方法的线程执行体，则该线程处于运行状态。如果计算机只有一个 CPU，那么在任何时刻只有一个线程处于运行状态，当然在一个多处理器的机器上，将会有多个线程并行执行；当线程数大于处理器数时，依然会存在多个线程在同一个 CPU 上轮换的现象。

当一个线程开始运行后，它不可能一直处于运行状态(除非它的线程执行体足够短，瞬间就执行结束了)。线程在运行过程中需要被中断，目的是使其他线程获得执行的机会，线程调度的细节取决于底层平台所采用的策略。对于采用抢占式策略的系统而言，系统会给每个可执行的线程一个小时间段来处理任务；当该时间段用完后，系统就会剥夺该线程所占用的资源，让其他线程获得执行的机会。在选择下一个线程时，系统会考虑线程的优先级。

所有现代的桌面和服务器操作系统都采用抢占式调度策略，但一些小型设备如手机则可能采用协作式调度策略。在这样的系统中，只有当一个线程调用了它的 sleep()或 yield()方法后才会放弃所占用的资源——也就是必须由该线程主动放弃所占用的资源。

当发生如下情况时，线程将会进入阻塞状态。

(1) 线程调用 sleep()方法主动放弃所占用的处理器资源。

(2) 调用了一个阻塞式 IO 方法,在该方法返回之前,该线程被阻塞。

(3) 线程试图获得一个同步监视器,但该同步监视器正被其他线程所持有。

(4) 线程在等待某个通知(notify)。

(5) 程序调用了线程的 suspend()方法将该线程挂起。但这个方法容易导致死锁,所以应该尽量避免使用该方法。

当前正在执行的线程被阻塞之后,其他线程就可以获得执行的机会。被阻塞的线程会在合适的时候重新进入就绪状态。注意,是就绪状态而不是运行状态。也就是说,被阻塞线程的阻塞解除后,必须重新等待线程调度器再次调度它。

针对上面几种情况,当发生如下特定的情况时可以解除上面的阻塞,让该线程重新进入就绪状态。

(1) 调用 sleep()方法的线程经过了指定时间。

(2) 线程调用的阻塞式 IO 方法已经返回。

(3) 线程成功获得了试图取得的同步监视器。

(4) 线程正在等待某个通知时,其他线程发出了通知。

(5) 处于挂起状态的线程被调用了 resdme()恢复方法。

3. 线程死亡

线程会以如下 3 种方式结束,结束后就处于死亡状态。

(1) run()或 call()方法执行完成,线程正常结束。

(2) 线程抛出一个未捕获的 Exception 或 Error。

(3) 直接调用该线程 stop()方法来结束该线程。该方法容易导致死锁,通常不推荐使用。

8.4 线程调度和线程优先级

8.4.1 线程的调度

在本章开头,我们提过计算机的 CPU 并不能让所有的任务同时执行,而是使用调度的方法分配时间给各个不同的进程来使用。而在进程内部也有这样的情况发生。

线程调度器选择优先级最高的线程运行。但是,如果发生以下情况,就会终止线程的运行。

(1) 线程体中调用了 yield()方法,让出了对 CPU 的占用权。

(2) 线程体中调用了 sleep()方法,使线程进入睡眠状态。

(3) 线程由于 I/O 操作而受阻塞。

(4) 另一个更高优先级的线程出现。

(5) 在支持时间片的系统中,该线程的时间片用完。

8.4.2 线程的优先级

线程的优先级是为了在多线程环境中便于系统对线程的调度,优先级高的线程将优

先执行。就像前面所说的,进程分配给线程的工作也是分轻重缓急的,处理不同紧急程度的事情也是我们作为开发者需要考虑的。

一个线程的优先级设置遵从以下原则。

(1) 线程创建时,子继承父的优先级。

(2) 线程创建后,可通过调用 setPriority()方法改变优先级。

(3) 线程的优先级为 1～10 的正整数。MIN_PRIORITY 为数字 1; MAX_PRIORITY 为数字 10; NORM_PRIORITY 为数字 5; 如果什么都没有设置,默认值是 5。

但是不能依靠线程的优先级来决定线程的执行顺序。

8.5　创建线程的两种方式

8.5.1　继承 Thread 类创建线程类

(1) 定义 Thread 类的子类,并重写该类的 run()方法,该 run()方法的方法体就代表了线程要完成的任务。因此把 run()方法称为执行体。

(2) 创建 Thread 子类的实例,即创建了线程对象。

(3) 调用线程对象的 start()方法来启动该线程。

下面是示例代码:

```
public class FirstThreadTest extends Thread{
    @Override
    public void run(){
        //在此处进行线程处理逻辑代码的编写
    }
    public static void main(String[] args)
    {
        new FirstThreadTest().start();
    }
}
```

8.5.2　通过 Runnable 接口创建线程类

(1) 定义 Runnable 接口的实现类,并重写该接口的 run()方法,该 run()方法的方法体同样是该线程的线程执行体。

(2) 创建 Runnable 实现类的实例,并依此实例作为 Thread 的 target 来创建 Thread 对象,该 Thread 对象才是真正的线程对象。

(3) 调用线程对象的 start()方法来启动该线程。

下面是示例代码:

```
public class RunnableThreadTest implements Runnable
{
    @Override
```

```
        public void run(){
                //在此处进行线程处理逻辑代码的编写
        }
        public static void main(String[] args)
        {
            RunnableThreadTest rtt = new RunnableThreadTest();
                    new Thread(rtt,"新线程 1").start();
        }
}
```

以上两种方法在使用上有所不同，在实际的使用中，应该结合实际编程情景来考虑对这两种方式进行线程处理逻辑的编写。

8.6 线程常用方法

(1) start()：线程调用该方法将启动线程，使之从新建状态进入就绪队列排队，一旦轮到它来享用 CPU 资源时，就可以脱离创建它的线程独立开始自己的生命周期了。

(2) run()：Thread 类的 run()方法与 Runnable 接口中的 run()方法的功能和作用相同，都用来定义线程对象被调度之后所执行的操作，都是系统自动调用而用户程序不得引用的方法。

(3) sleep(int millsecond)：优先级高的线程可以在它的 run()方法中调用 sleep()方法来使自己放弃 CPU 资源，休眠一段时间。

(4) isAlive()：线程处于“新建”状态时，线程调用 isAlive()方法返回 false。在线程的 run()方法结束之前，即没有进入死亡状态之前，线程调用 isAlive()方法返回 true。

(5) currentThread()：该方法是 Thread 类中的类方法，可以用类名调用，该方法返回当前正在使用 CPU 资源的线程。

(6) interrupt()：一个占有 CPU 资源的线程可以让休眠的线程调用 interrupt()方法“吵醒”自己，即导致休眠的线程发生 InterruptedException 异常，从而结束休眠，重新排队等待 CPU 资源。

8.7 线程同步

8.7.1 线程同步理解

究竟什么是线程同步呢？在了解线程同步之前，我们先了解一下什么是同步，这有助于对线程同步的理解。所谓同步，就是在发出一个方法的调用时，在没有得到结果之前，这个调用就不返回，同时其他的线程也不能调用这个方法！线程同步也是类似的意思，但线程同步不是说让一个线程执行完了再执行其他线程，一般是指让线程中的某一些操作进行同步就可以了。

在多线程的编程里，我们不可避免地会遇上这样的一种问题，一些数据不能被多个线

程同时访问，比如A和B同时去商店里买糖，但商店里一共就只剩下3颗糖了，A说我要2颗，B同时也说要2颗，那么此时是不是A和B中肯定有一个人买不到2颗糖！如果生活中还好，还可以商量着解决！那么线程里有没有类似商量解决的方法呢？其实多线程里面也是有类似方法的，先让A或B中的一个买，假设是A先买，等A先买完了，B再买！同步机制就可以解决上面这个问题，解决让谁先买，谁后买的问题。采用同步机制可以保证数据在任何时候最多只有一个线程进行访问，从而保证了数据的安全！

等A和B处理完买糖的事之后，他们就可以想干嘛就干嘛了。所以说线程同步，一般是指让线程中的某一些操作进行同步就可以了。

8.7.2 线程同步实现

(1) 在需要同步的函数的函数签名中加上synchronized关键字。

(2) 使用synchronized关键字对需要进行同步的代码块进行同步。

(3) 使用java.util.concurrent.lock包中Lock对象(JDK1.5以上)。

注意：

(1) synchronized是对当前的实例进行加锁，要注意是“当前实例”，也就是说，假如你有两个实例化对象，那么可以同时访问这两个实例里面的synchronized块。但是，当访问一个实例里面的一个synchronized块时，其余的synchronized是不可同时访问的，原因是整个实例都被加了锁。

(2) synchronized关键字是不能够继承的。

8.8 计时器 Timer

在开发中，我们经常需要一些周期性的操作，例如每隔几分钟就进行某一项操作。这时候我们就要去设置定时器，Java中最方便、最高效的实现方式是用java.util.Timer工具类，再通过调度java.util.TimerTask执行任务。

Timer是一种工具，线程用其安排以后在后台线程中执行的任务。可安排任务执行一次，或者定期重复执行。实际上是个线程，定时调度所拥有的TimerTasks。

TimerTask是一个抽象类，它的子类由Timer安排为一次执行或重复执行的任务。实际上就是一个拥有run()方法的类，需要定时执行的代码放到run()方法体内。

在工具类Timer中，提供了四个构造方法，每个构造方法都启动了计时器线程，同时Timer类可以保证多个线程可以共享单个Timer对象而无须进行外部同步，所以Timer类是线程安全的。但是由于每一个Timer对象对应的是单个后台线程，用于顺序执行所有的计时器任务。一般情况下我们的线程任务执行所消耗的时间应该非常短，但是由于特殊情况导致某个定时器任务执行的时间太长，那么它就会“独占”计时器的任务执行线程，其后的所有线程都必须等待它执行完，这就会延迟后续任务的执行，使这些任务堆积在一起。

当程序初始化完成Timer后，定时任务就会按照我们设定的时间去执行，Timer提供了schedule()方法，该方法有多种重载方式来适应不同的情况，如下所示。

schedule(TimerTask task, Date time):安排在指定的时间执行指定的任务。

schedule(TimerTask task, Date firstTime,long period):安排指定的任务在指定的时间开始进行重复的固定延迟执行。

schedule(TimerTask task,long delay):安排在指定延迟后执行指定的任务。

schedule(TimerTask task,long delay,long period):安排指定的任务从指定的延迟后开始进行重复的固定延迟执行。

同时也重载了 scheduleAtFixedRate 方法,scheduleAtFixedRate 方法与 schedule 相同,只不过它们的侧重点不同,其区别将在后面分析。

scheduleAtFixedRate(TimerTask task,Date firstTime,long period):安排指定的任务在指定的时间开始以固定速率进行重复的执行。

scheduleAtFixedRate(TimerTask task,long delay,long period):安排指定的任务在指定的延迟后开始进行以固定速率重复的执行。

TimerTask 类是一个抽象类,由 Timer 安排为一次执行或重复执行的任务。它有一个抽象方法,即 run()方法,该方法用于执行相应计时器任务要执行的操作。因此每一个具体的任务类都必须继承 TimerTask,然后重写 run()方法。

另外它还有两个非抽象的方法,如下所示。

(1) boolean cancel():取消此计时器任务。

(2) long scheduledExecutionTime():返回此任务最近实际执行的安排执行时间。

下面是示例代码:

```
Timer timer = Timer(true);
//注意,javax.swing包中也有一个 Timer类,如果 import中用到 swing包,要注意名字的
冲突
TimerTask task = new TimerTask() {
    public void run() {
    ... //每次需要执行的代码放到这里面
    }
};

//以下是几种常用调度 task的方法

timer.schedule(task, time);
//time为 Date类型:在指定时间执行一次

timer.schedule(task, firstTime, period);
//firstTime为 Date类型,period为 long
//从 firstTime时刻开始,每隔 period毫秒执行一次

timer.schedule(task, delay)
//delay为 long类型:从现在起过 delay毫秒执行一次

```

```
timer.schedule(task, delay, period)
//delay为 long,period为 long:从现在起过 delay毫秒以后,每隔 period
//毫秒执行一次
```

8.9　图书管理系统 V8.0

我们只给出了服务器端的部分,剩下部分请同学们自行完成。

8.9.1　运行效果图

服务端的运行效果图和第 7 节服务器端效果图一致。

8.9.2　类结构示意图

图 8-2 和图 8-3 分别展示了服务器端类结构和服务器端类 MVC 三层结构。

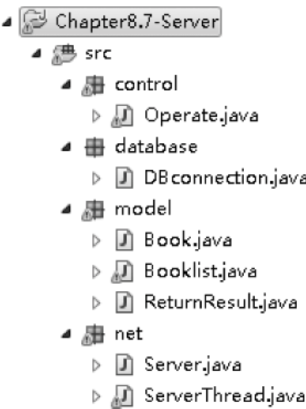


图 8-2　服务器端类结构图

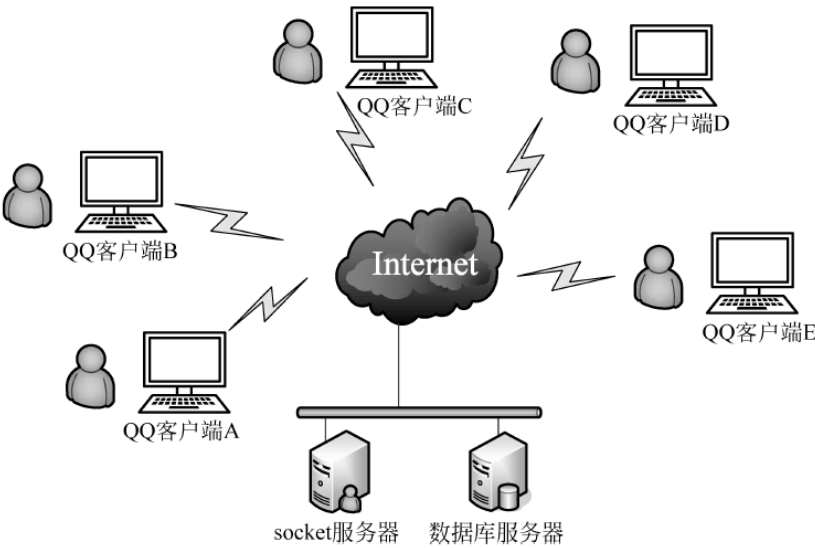


图 8-3　服务器端类 MVC 三层结构图



8.9.3 代码实现

服务器端只修改了 Server.java 类，修改成 Server.java 类和 serverThread.java 类，具体代码如下：

Server.java

```
package net;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class Server {

    public Server() {
        ServerSocket ss = null;
        try {
            ss = new ServerSocket(1234);
            while (true) {
                System.out.println("服务器开启了，等待连接…");
                Socket tcpConnection = ss.accept();

                ServerThread ct = new ServerThread(tcpConnection);
                ct.start();
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        new Server();
    }

}
```

ServerThread.java

```
package net;

import java.io.IOException;
```

```
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
import control.Operate;
import model.Book;
import model.ReturnResult;

public class ServerThread extends Thread {
    Socket tcpConnection;
    byte[] buffer = new byte[256];

    public ServerThread(Socket tcpConnection) {
        this.tcpConnection = tcpConnection;
    }

    public void run() {
        try {
            System.out.println("连接上客户端了，客户端 IP:" + tcpConnection.
            getInetAddress() + ",端口为:" + tcpConnection.getPort());
            InputStream in = tcpConnection.getInputStream();
            OutputStream out = tcpConnection.getOutputStream();
            while (true) {
                int count = in.read(buffer);
                String str = new String(buffer, 0, count);
                if (str.equals("closeserver")) {
                    break;
                }
                String[] totalstr = str.split("/n");
                String head = totalstr[0];
                String tail = totalstr[1];
                if (head.equals("operate:add")) {
                    Operate operator = new Operate();
                    String[] strArray = null;
                    strArray = tail.split(",");
                    String bookname = strArray[0];
                    String author = strArray[1];
                    String aprice = strArray[2];
                    float price = Float.parseFloat(aprice);
                    Book book = new Book(bookname, author, price);
                    ReturnResult returnresult = new ReturnResult();
```

```
                    returnresult = operator.addBook(book);
                    if (returnresult.isSuccess()) {
                        out.write("result:Success".getBytes());
                    } else {
                        out.write("result:notSuccess".getBytes());
                    }
                }
                if (head.equals("operate:delete")) {
                    Operate operator = new Operate();
                    ReturnResult returnresult = new ReturnResult();
                    returnresult = operator.deletebook(tail);
                    if (returnresult.isSuccess()) {
                        out.write("result:Success".getBytes());
                    } else {
                        out.write("result:notSuccess".getBytes());
                    }
                }

            }
            out.write("close".getBytes());
            in.close();
            out.close();
            tcpConnection.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
}
```

第 9 章

扩展——从图书管理系统到 QQ

阅读至此，我们已经将 Java 核心基础的内容讲解完，相信各位读者对于整个 Java 编程语言都有了自己的认识。养兵千日用在一时，我们将在这一章节对本书讲解的知识点进行整合，自主设计和开发一个简易版的 QQ。

9.1 本章任务

理论任务：对本书中所讲解的 Java 基础知识点，如类、对象、多线程、数据存储、网络编程等，巩固基础并深入理解。

实践任务：结合本书中所学习的 Java 编程核心概念以及语法，我们将完成一个模拟“QQ”的通信交友软件的设计和开发。当然，实际 QQ 的功能、软件架构要复杂得多，希望起到抛砖引玉的作用。

9.2 总体结构

在本书中所学习到的知识点，将用于开发一个简易版的 QQ 程序。首先，需要明确“QQ”需要做到客户端与服务器分离，并且需要使用 TCP 协议来连接服务器与客户端，并且需要制定每个功能的通信协议或解析方式。其次，需要采用合理的设计模式对软件进行设计，这样有助于提升效率，更清晰地编写出各部分代码，各功能分离清晰。最后，服务器端也会需要使用数据库对用户的数据进行存储，并且使用多线程服务多个用户，并将上线用户的地址和线程存入 Map 中，形成用户池，当 A 用户向 B 用户发送请求时，服务器便可从用户池中将 B 用户对应的线程取出，并将 A 用户发送的消息解析后发送给 B 用户，B 用户的客户端解析客户端发送的数据，提示 B 用户查看，完成一个消息的发送和接收过程。若有兴趣实现其他功能，也可自己进行设计和开发。图 9-1 为整个 QQ 的总体结构。

由总体结构图我们就会发现，我们想要实现类 QQ 即时通讯的功能其实并不困难，图 9-1 所给出的也就是整个 QQ 从用户相互之间进行沟通的最为简单的框架结构。每个独立的客户端在登录之后，服务器通过验证客户端发送的用户信息，来存储用户的信息到内存，同时也就可以将该用户的好友列表和在线情况发送给该用户所在的客户端。

当某个用户通过客户端向另一名用户发送信息时，服务器便可从内存中读取目标用户信息，从而将用户发送的消息转发给目标用户。当然，如果你想要在原来的基础之上对

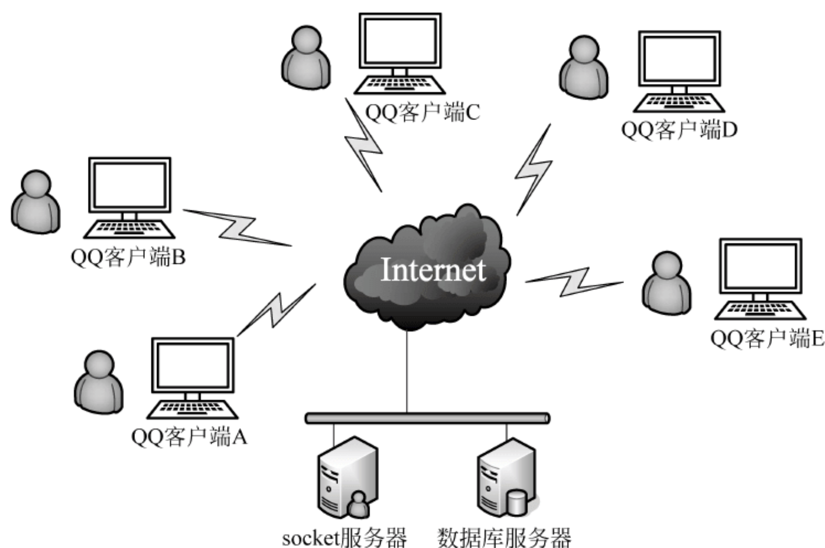


图 9-1　QQ 总体结构图

QQ 最为简单的通信功能进行升级，那么你也可以通过扩充服务器的逻辑代码和存储信息的方式来完成，诸如离线消息、消息提示功能的开发。

9.3　服务器端

在本节，我们将讲解如何完成 QQ 服务器端的开发。其实，在开发图书管理系统 V7.0时，我们就已经了解到了服务器的作用和监听端口以及打开获取连接一系列的操作是如何通过代码来实现的。

首先，开发 QQ 服务器需要考虑以下几点。

（1）数据库的使用——主要用来存储用户的账号密码、用户的聊天记录。

（2）通信协议——主要用来规范每个用户操作，发送消息后，服务器能够统一进行解析，并拆分处理。

（3）在线用户池管理——主要使用 HashMap < String，Thread >这样一个 HashMap 对上线用户的线程进行存储。

其次，在解决上述几点需要注意的问题之后，我们需要把重心放到理解在线用户池的管理这一具体的问题上。为什么呢？因为这是 QQ 实现不同用户之间进行通信的基础。整个通信过程应该是这样的：在 A、B 用户在线的情况下，当 A 用户打开与 B 用户的聊天框并发送一条消息后，服务器对 A 用户发送的消息进行处理，得知需要将 A 用户发送的消息转发给 B 用户后，服务器则在在线用户池中，将 B 用户服务器线程的引用通过 B 用户的昵称取出。并将 A 用户发送的消息通过 B 用户的服务器线程引用发送给 B 用户。反之，则是服务器通过查找 A 用户的服务器线程引用将 B 用户发送的消息转发给 A 用户。

现在，整个服务器的功能处理流程都较为清晰了。用户在进行登录验证后，服务器会将当前用户所使用的服务器线程引用放入线程池。当用户发起对另一个在线用户的对话

时，服务器收到消息后，则会将这一用户的线程引用从线程池中取出，而后将消息通过线程进行转发。这样服务器就完成了整个 QQ 最基本功能——在线用户的聊天。

以下是 QQ 服务器的代码框架，只提供了关键的登录以及发送消息的处理逻辑代码，诸如注册、添加好友等代码均未给出，同学们可以参照这一框架来对代码进行完善，也可以按照自己的理解进行编写，完成后，再对代码进行优化和重构，达到熟悉 Java 代码知识点的目的。

9.3.1 运行效果图

完成本章的开发任务之后，服务器的运行截图如图 9-2 所示。当然，实际的 QQ 软件服务器后台管理软件要复杂得多，比如可以对所有用户的管理等。

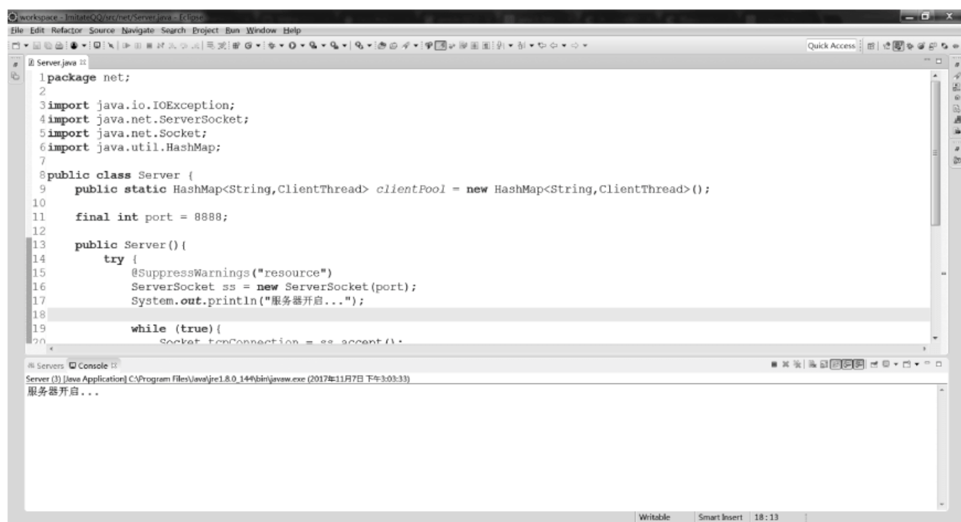


图 9-2　QQ 总体结构图

9.3.2 类结构示意图

1. 类结构

图 9-3 展示了类的结构。



图 9-3　类结构图

2. 代码逻辑结构

QQ 服务器的大致工作原理和内部的部分细节如图 9-4 所示，我们从中可以了解到整个 QQ 的工作流程，对于实现这一服务器有很大的帮助。

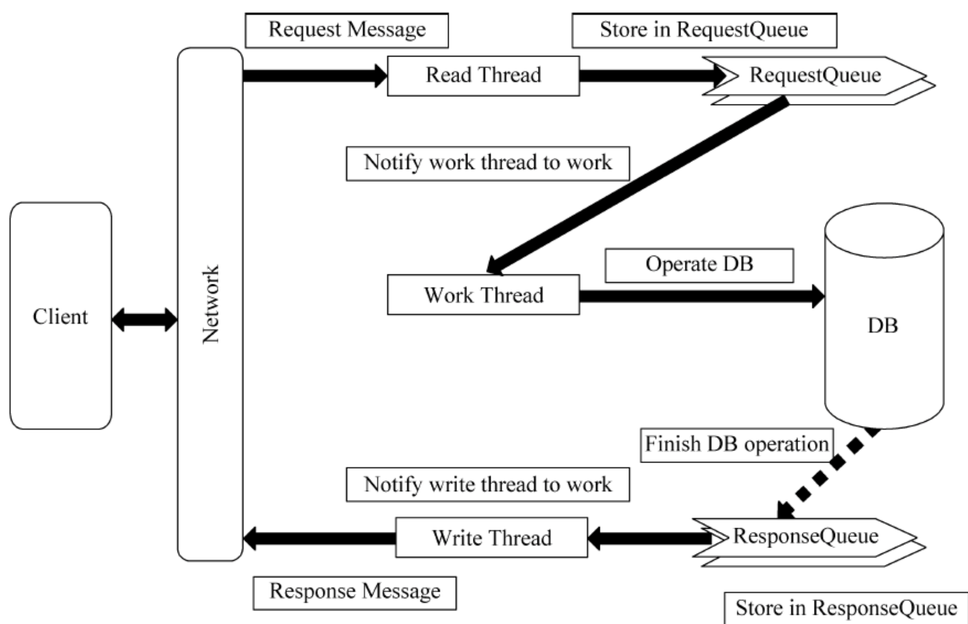


图 9-4 QQ 服务器端代码逻辑图

9.3.3 代码实现

在本章的开发中，我们只给出一部分关键代码，例如如何开启服务器线程和登录协议的判断等等。其他诸如数据连接等基本类和工具类均不给出，读者在自行编写时若遇到困难，返回本书前几章查看相关代码即可。

服务器主类，用于监听端口获取用户 IP 和开启子用户线程。

Serve.java

```
package net;

import java.io.IOException;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.HashMap;

public class Server {
    //使用 HashMap 对上线用户进行管理
    public static HashMap<String,ClientThread> clientPool = new
    HashMap<String,ClientThread>();

```

```
    final int port = 8888;

    public Server(){
        try {
            @SuppressWarnings("resource")
            ServerSocket ss = new ServerSocket(port);
            System.out.println("服务器开启...");

            while (true){
                Socket tcpConnection = ss.accept();
                System.out.println("IP地址为:" + tcpConnection.
                getInetAddress().getHostAddress() +"的用户打开客户端");
                ClientThread ct = new ClientThread(tcpConnection);
                ct.start();
            }

        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        new Server();

    }
}
```

服务器子线程类，用于监听和处理客户端发送的请求。

ClientThread.java

```
package net;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
import operator.Operation;
import util.Protocol;

public class ClientThread extends Thread{
    Socket tcpConnection;
```




```

byte[] buffer =  new byte[256];

byte[] bufferString = new byte[256];

Operation operation = new Operation();

InputStream in;

OutputStream out;

Protocol protocol = new Protocol();

public ClientThread(Socket tcpConnection){
    this.tcpConnection = tcpConnection;
}

public void start(){
    try {
        in = tcpConnection.getInputStream();
        out = tcpConnection.getOutputStream();

        while (true)
        {
            int count = in.read(buffer);
            String receiveString = new String(buffer,0,count);
            System.out.println(receiveString);
            //对收到的内容进行解析
            String[] massagecontent = receiveString.split(":");
            String head = massagecontent[0];
            String massage = massagecontent[1];

            if (head.equals("LogIn"))
            {
                String[] content = massage.split(",");
                String username = content[0];
                String password = content[1];

                if(operation.checkloginaccount(username, password))
                {
                    String account = username;
                    //登录成功后将用户线程放入 HashMap 中保存
                    Server.clientPool.put(account, this);
                    String friendID =operation.returnfriedlist(username);
```

```
                    String finalmassage = "success;" +friendID ;
                    this.sendtoClient(finalmassage);
                }
                else{
                    sendtoClient("fail");
                }
            }

            if (head.equals("SendMassage"))
            {
                String[] content = massage.split(",");
                String from = content[0];
                String to = content[1];
                String conversation = content[2];
                String[] afrom = from.split(";");
                String postername = afrom[1];
                String[] ato = to.split(";");
                String receiver = ato[1];
                String postmassage = protocol.postFriendmassage(postername,
                conversation,receiver);
                        System.out.println(postmassage);
                ClientThread toClient = Server.clientPool.get(receiver);
                //从池中将相应的线程取出,并将消息发送至目标用户
                //若目标用户离线,不作处理,用户可自行添加离线消息功能
                if(toClient!=null){
                    toClient.sendtoClient(postmassage);
                }
            }
        }
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

public void sendtoClient(String message)
{
    try
    {
        out.write(message.getBytes());
    }
    catch (IOException e)
    {
        e.printStackTrace();
```

```
            }
        }
    }
```

协议类用于打包发送给客户端的消息，将其分装为固定的格式。

协议格式为：消息头（功能名称）：消息体 1 ：消息体 2 ：… ：消息体 n；

Protocol.java

```
package util;

public class Protocol
{
    public Protocol()
    {

    }

    public String postFriendmassage(String postername,String postmassage,String nickname)
    {
        String massage = "Newmassage:" + postername + ":" + postmassage+ ":" + nickname;
        return massage;
    }

    public String postFriendRequest(String nickname,String friendname)
    {
        String massage = "AddFriendrequest:" + nickname + ":" + friendname;return massage;
    }
}


```

9.4 客　户　端

在本节，我们将讲解如何完成 QQ 客户端的开发。QQ 客户端主要的作用是发送消息和接收消息。与图书管理系统网络版类似，客户端需要在打开后，与服务器进行连接，并且实时记录用户输入的信息，对消息进行收发操作。

首先，开发 QQ 客户端需要考虑以下几点。

(1) 通信协议——主要用来规范每个用户操作，发送消息后，服务器能够统一进行解析，并拆分处理。

(2) 用户对话窗口池管理——主要使用 HashMap < String,JFrame >这样一个 HashMap 对用户的聊天窗口进行管理。

(3) 后台线程——主要用来在后台无限循环监听接收服务器的消息。

其次，在解决上述几点需要注意的问题之后，我们需要把重心放到理解用户对话窗口池的管理这一具体的问题上。为什么呢？实现窗口池的作用大致与服务器端的用户窗口池相似。当 A 用户和 B 用户同时在线，A 用户向 B 用户发送消息，但这时，后台进程并不知晓 B 用户打开了与 A 用户聊天的窗口，这时后台线程需要对此情形进行判断，如果 B 用户打开了聊天窗口，那么在窗口池中，我们便可以获取到聊天窗口的引用，通过取出引用从而将 A 用户发送的消息打印至聊天窗口中。当 B 用户没有打开与 A 用户的聊天窗口时，后台线程则会自动为 B 用户创建该聊天窗口并且完成消息的显示。这样 QQ 客户端最为基础的消息收发功能便已经完成。

现在，我们整个客户端的处理信息和显示信息的流程都较为清晰了。以下是 QQ 客户端的代码框架，只提供了关键的后台线程、聊天窗口等处理逻辑代码，诸如注册、添加好友等代码均未给出，同学们可以参照这一框架来对代码进行完善，也还可以按照自己的理解进行编写，完成后，再对代码进行优化和重构，达到熟悉 Java 代码知识点的目的。

9.4.1 运行效果图

完成开发后，QQ 的登录界面如图 9-5 所示，注册页面如图 9-6 所示，好友列表界面如图 9-7 所示，聊天界面如图 9-8 所示。这里只展示关于登录界面和注册界面的部分运行效果截图，对于聊天界面同学们可以自行发挥想象力设计开发，漂亮的界面更是能够增加同学们对于编程的热情。



图 9-5　QQ 登录界面运行截图



图 9-6　QQ 注册界面运行截图

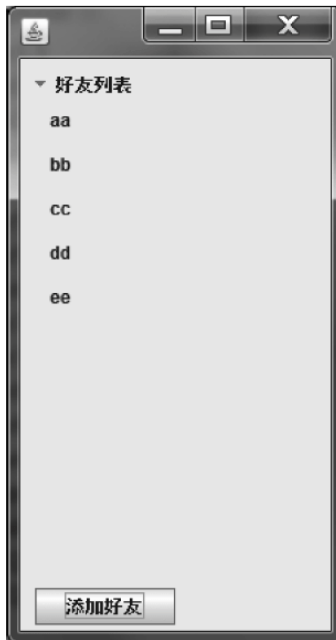


图 9-7　QQ 好友列表界面运行截图

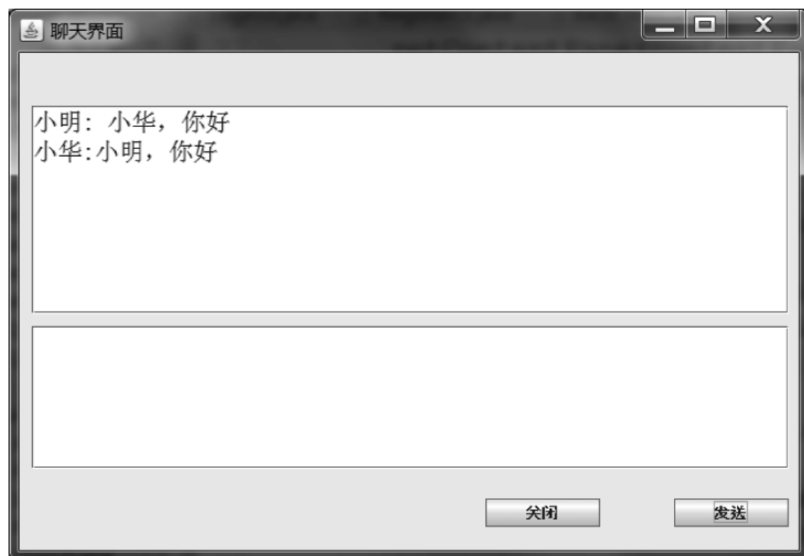


图 9-8　聊天界面运行截图

9.4.2 类结构示意图

1. 类结构

图 9-9 展示了类的结构。

2. 代码逻辑结构

QQ 客户端的大致工作原理和内部的部分细节如图 9-10 所示，从中我们可以了解到

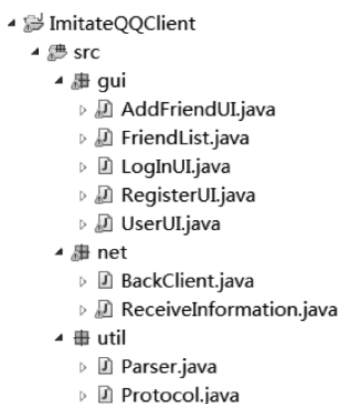


图 9-9　类结构图

整个 QQ 客户端的工作流程,对于实现客户端有很大的帮助。主要由 HashMap 在其中起到接到消息后的查找用户聊天窗口的作用。

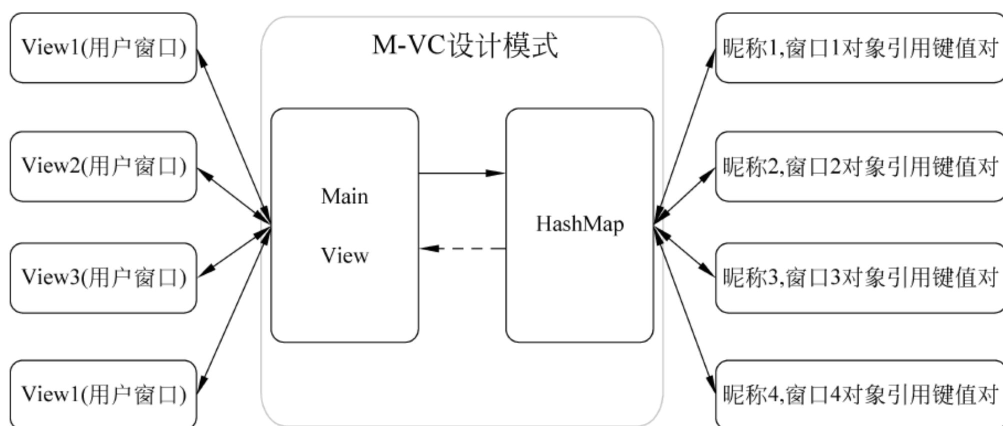


图 9-10　QQ 工作原理图

9.4.3　代码实现

在本章的开发中,我们同样只给出一部分关键代码,例如登录、注册和好友列表界面的相关代码以及后台线程如何监听和创建 HashMap 来保存 UI 的具体代码等。其他诸如数据连接等基本类和工具类均不给出,读者在自行编写时若遇到困难,返回本书前几章查看相关代码即可。

登录界面窗口代码的主要作用在于显示输入框和交互界面、获取用户在 UI 中输入的数据和使用后台线程发送消息。

LogInUI.java

```
package gui;

import java.awt.EventQueue;
import javax.swing.JFrame;
```




```
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import java.awt.GridLayout;
import javax.swing.JButton;
import java.awt.Font;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.JPasswordField;
import util.Parser;
import util.Protocol;
import net.BackClient;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
public class LogInUI extends JFrame
{
    private static final long serialVersionUID = 7639002619638253931L;

    final int num = 521;
    private JPanel contentPane;

    private JTextField userField;

    private JPasswordField passwordField;

    byte[] sentence = new byte[num];

        BackClient backClient = new BackClient();

        Protocol protocol = new Protocol();

        Parser parser = new Parser();
        /**
        * Launch the application.
         */
        public static void main(String[] args)
        {
            EventQueue.invokeLater(new Runnable()
            {
                public void run()
                {
                    try
                    {
                        LogInUI frame = new LogInUI();
```

```
                    frame.setVisible(true);
                    frame.setResizable(false);
                } catch (Exception e)
                {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
    public LogInUI()
    {
        setTitle("登录");
        getContentPane().setLayout(new GridLayout(1, 0, 0, 0));
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 304, 162);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JLabel label = new JLabel("账号:");
        label.setFont(new Font("宋体", Font.PLAIN, 17));
        label.setBounds(20, 29, 51, 24);
        contentPane.add(label);

        JLabel label_1 = new JLabel("密码:");
        label_1.setFont(new Font("宋体", Font.PLAIN, 17));
        label_1.setBounds(20, 63, 56, 25);
        contentPane.add(label_1);

        userField = new JTextField();
        userField.setBounds(81, 32, 78, 21);
        contentPane.add(userField);
        userField.setColumns(10);

        passwordField = new JPasswordField();
        passwordField.setBounds(81, 67, 78, 21);
        contentPane.add(passwordField);

        JButton button = new JButton("登录");
```




```
button.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent arg0)
    {
        String nickname = userField.getText();
        String password = String.valueOf(passwordField.getPassword());
        String information = nickname + "," + password;

        String massage = protocol.getLogInInformation(information);
        backClient.SendMassage(massage);
        String recieve = backClient.ReseiveMassage();

        System.out.println(recieve);
        String[] temp = recieve.split(";");
        String head = temp[0];
        boolean issuccess = parser.isLogin(head);
        if(issuccess)
        {
            String friendID = temp[1];
            LogInUI.this.setVisible(false);;
            FriendList ui = new FriendList();
            ui.setVisible(true);
            ui.setFirendID(friendID);
            ui.setNickname(nickname);
        }
        else{
            JOptionPane.showMessageDialog(null, "账号密码错误");
        }
    }
});
button.setBounds(187, 65, 95, 25);
contentPane.add(button);

JLabel lblNewLabel = new JLabel("登录");
lblNewLabel.setFont(new Font("宋体", Font.PLAIN, 15));
lblNewLabel.setBounds(122, 7, 62, 15);
contentPane.add(lblNewLabel);

JButton button_1 = new JButton("注册");
button_1.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent arg0)
    {
        RegisterUI registerUI = new RegisterUI();
```

```
                    registerUI.setVisible(true);
                    LogInUI.this.setVisible(false);
                }
            });
            button_1.setBounds(187, 30, 95, 25);
            contentPane.add(button_1);
        }
    }
```

好友列表代码，主要用于登录成功后显示服务器返回的好友昵称信息，并将其按照预先设计好的排版罗列在 UI 界面中，当需要和某同学聊天时，通过单击昵称弹出对话框进行对话和信息的收发显示。

FriendList.java

```
package gui;
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel;
import javax.swing.ImageIcon;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JButton;
import util.Protocol;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import net.BackClient;
import net.ReceiveInformation;

public class FriendList extends JFrame {

    private JPanel contentPane;

    String nickname;

    String friendnickname ;

    JLabel []list = new JLabel[10];

    String[] friendID;

    ReceiveInformation information = new ReceiveInformation();

    Thread thread = new Thread(new ReceiveInformation());
```

```
    int count;

    Protocol protocol = new Protocol();

    BackClient backClient = new BackClient();

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    FriendList frame = new FriendList();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    public FriendList() {
        information.addthisWindows("friendlist", this);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 228, 433);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        final JLabel lblNewLabel = new JLabel("好友列表");
        lblNewLabel.setIcon(new ImageIcon("3.png"));
        lblNewLabel.setBounds(10, 10, 192, 15);
        contentPane.add(lblNewLabel);
        JButton button = new JButton("添加好友");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                AddFriendUI addFriendUI = new AddFriendUI();
                addFriendUI.getNickname(nickname);
                addFriendUI.setVisible(true);
            }
        });
        button.setBounds(10, 359, 95, 25);
        contentPane.add(button);
```

```
        thread.start();

        lblNewLabel.addMouseListener(new MouseAdapter()
        {
            int isdrag = 1;
            @Override
            public void mouseClicked(MouseEvent e)
            {
                if(e.getClickCount()==1 )
                {
                    if(isdrag%2==0)
                    {
                        lblNewLabel.setIcon(new ImageIcon("3.png"));
                        for(int i=0;i<friendID.length;i++)
                        {
                            list[i].setVisible(true);
                        }
                    }
                    if(isdrag%2==1)
                    {
                        lblNewLabel.setIcon(new ImageIcon("aa.png"));
                        for(int i=0;i<friendID.length;i++)
                        {
                            list[i].setVisible(false);
                        }
                    }
                    isdrag++;
                }
            }
        });
    }

    public void setNickname(String nickname)
    {
        this.nickname = nickname;
        String offlinemessage = protocol.getOfflineMassage(nickname);
        backClient.SendMassage(offlinemessage);
        String okmassage = protocol.backOfflineMassageOk(nickname);
        backClient.SendMassage(okmassage);
    }

    public void setFirendID(String friendnickname)
    {
        this.friendnickname = friendnickname;
```

```
        friendID = friendnickname.split(",");

        final String[] id = new String[50];

        for(int i=0;i<friendID.length;i++)
        {
            list[i] = new JLabel(friendID[i]);

            final String aid = friendID[i];
            list[i].setBounds(20, 30 * (i+1), 182, 22);
            contentPane.add(list[i]);
            list[i].addMouseListener(new MouseAdapter()
            {
                @Override
                public void mouseClicked(MouseEvent e)
                {
                    if(e.getClickCount()==2)
                    {
                        UserUI ui = new UserUI();
                        ui.getfrinedID(aid);
                        ui.setNickname(nickname);
                        ui.setVisible(true);
                    }
                }
            });
        }
        count = friendID.length;
    }

    public void setNewFriend(String friendID)
    {
        list[count] = new JLabel(friendID);
        list[count].setBounds(20, 30 * (count+1), 182, 22);
        contentPane.add(list[count]);

        final String aid = friendID;
        list[count].addMouseListener(new MouseAdapter()
        {
            @Override
            public void mouseClicked(MouseEvent e)
            {
                if(e.getClickCount()==2)
                {
                    UserUI ui = new UserUI();
```

```
                        ui.getfrinedID(aid);
                        ui.setNickname(nickname);
                        ui.setVisible(true);
                    }
                }
            });
            contentPane.repaint();
        }
}
```

收取信息主类，主要用于在后台使用监听端口获取服务器发送的信息，并对其信息进行处理，随后做出相应的操作，对用户的 UI 进行修改和通知。

ReceiveInformation.java

```
package net;

import gui.FriendList;
import gui.UserUI;
import java.util.HashMap;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import util.Protocol;

public class ReceiveInformation implements Runnable
{
    boolean isok;

    static HashMap<String,JFrame> clientPool;

    Protocol protocol = new Protocol();

    static
    {
        clientPool = new HashMap<String,JFrame>();
    }
    BackClient backClient = new BackClient();

    public ReceiveInformation()
    {
    }

    public void addthisWindows(String windowname,JFrame jframe)
    {
        JFrame thisframe = clientPool.put(windowname,jframe);
    }
```

```

public JFrame returnthisWindows(String windowname)
{
    JFrame thisframe = clientPool.get(windowname);
    return thisframe;
}

public boolean addFriendIsok()
{
    return isok;
}

public void run()
{
    String str = backClient.ReseiveFriendMassage();
    String[] content = str.split(":");
    String head = content[0];

    if(head.equals("AddFriendrequest"))
    {
        String[] name = content[1].split(",");
        String friendID = name[0];
        String nickname = name[1];

        int choose = JOptionPane.showConfirmDialog(null,friendID+"想加您为好友");
        if(choose == JOptionPane.YES_OPTION)
        {
            String addmassage = protocol.getAddFriendOkMessage(nickname, friendID);
            backClient.SendMassage(addmassage);
            FriendList friendList = (FriendList)returnthisWindows ("friendlist");
            friendList.setNewFriend(friendID);
        }
        if(choose == JOptionPane.NO_OPTION)
        {
            String addmassage = protocol.getAddFriendNoMessage(nickname, friendID);
            backClient.SendMassage(addmassage);
        }
    }

    if(head.equals("NotExist"))
    {
        FriendList friendList = (FriendList)returnthisWindows("friendlist");
        JOptionPane.showMessageDialog(null,"查询好友不存在");
    }
```

```
            if(head.equals("Refuse"))
            {
                JOptionPane.showMessageDialog(null,"对方拒绝添加您为好友");
            }

            if(head.equals("addFriendok"))
            {
                String[] name = content[1].split(",");
                String friendID = name[0];
                String nickname = name[1];
                System.out.println(nickname+173);

                String addmassage = protocol.getAddFriendOkMessage(nickname, friendID);
                backClient.SendMassage(addmassage);
                JOptionPane.showMessageDialog(null,"发送成功");
                FriendList friendList = (FriendList)returnthisWindows("friendlist");
                friendList.setNewFriend(friendID);
            }
            if(head.equals("Newmassage"))
            {
                String friendID = content[1];
                String context = content[2];
                String nickname = content[3];
                UserUI frame = (UserUI) returnthisWindows(friendID);
                if(frame==null)
                {
                    UserUI ui = new UserUI();
                    ui.getfrinedID(friendID);
                    ui.setNickname(nickname);
                    ui.setVisible(true);
                    ui.setConversation(friendID,context);
                    addthisWindows(friendID,ui);
                }
                else
                {
                    frame.setConversation(friendID, context);
                }
            }
        }
    }
```

聊天窗口类主要用于显示向用户发送消息时的消息记录，通过收取消息类的监听并解析打印消息至聊天窗口 UI 中，使用户得以查看对方发送的消息。

UserUI.java

```
package gui;

import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JTextField;
import net.BackClient;
import net.ReceiveInformation;
import util.Parser;
import util.Protocol;
import javax.swing.JLabel;
import java.awt.Font;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class UserUI extends JFrame {

    /**
     *
     */
    private static final long serialVersionUID = 7331169995258952629L;

    private JPanel contentPane;

    BackClient backClient = new BackClient();

    Protocol protocol = new Protocol();

    Parser parser = new Parser();

    String nickname;

    String friendnickname;

    private JTextField textField;

    JTextArea textArea;

    Thread thread = new Thread(new ReceiveInformation());

    ReceiveInformation information = new ReceiveInformation();
```

```
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    UserUI frame = new UserUI();
                    frame.setVisible(true);

                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    public UserUI()
    {
        thread.start();
        setTitle("聊天界面");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 650, 449);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JScrollPane scrollPane = new JScrollPane();
        scrollPane.setBounds(10, 43, 614, 169);
        contentPane.add(scrollPane);

        textArea = new JTextArea();
        textArea.setEditable(false);
        Font f = new Font("宋体", Font.LAYOUT_LEFT_TO_RIGHT, 20);
        textArea.setFont(f);
        scrollPane.setViewportView(textArea);

        JButton sendbutton = new JButton("发送");
        sendbutton.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent arg0)
            {
                String friendname = friendnickname;
                String conversation = textField.getText();
                String from = "From;" +nickname + ",";
                String to = "To;" + friendname + ",";
                String massage = from + to + conversation;
```

```

                    textArea.append(nickname + ":" + conversation + "\n");
                    textField.setText("");

                    String finalmassage = protocol.getToFriendmassage(massage);
                    backClient.SendMassage(finalmassage);
                }
            });
            sendbutton.setBounds(531, 362, 93, 23);
            contentPane.add(sendbutton);

            JButton exitNewButton = new JButton("关闭");
            exitNewButton.addActionListener(new ActionListener()
            {
                public void actionPerformed(ActionEvent arg0)
                {
                    backClient.SendMassage("close");
                    setVisible(false);
                }
            });
            exitNewButton.setBounds(378, 362, 93, 23);
            contentPane.add(exitNewButton);

            textField = new JTextField();
            textField.setBounds(10, 222, 614, 116);
            contentPane.add(textField);
            textField.setColumns(10);

            contentPane.setVisible(true);
        }

        public void setNickname(String nickname)
        {
            this.nickname = nickname;
        }

        public void getfrinedID(String friendID)
        {
            this.friendnickname = friendID;
            information.addthisWindows(friendnickname, this);
            JLabel lblNewLabel_1 = new JLabel(friendnickname);
            lblNewLabel_1.setFont(new Font("宋体", Font.BOLD, 20));
            lblNewLabel_1.setBounds(10, 18, 54, 15);
            contentPane.add(lblNewLabel_1);
            lblNewLabel_1.setVisible(true);
        }
```

```java

    public void setConversation(String name,String content)
    {
        textArea.append(name + ":" +content + "\n");
    }
}
```

解析类主要用于解析服务器发送的信息，并按照规定的协议格式进行解析。具体的协议格式与服务器端的协议格式一致。

Paser.java

```java
package util;

public class Parser
{
    public Parser()
    {
    }

    public boolean isLogin(String massage)
    {
        if(massage.equals("success"))
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    public boolean isSent(String massage)
    {
        if(massage.equals("success"))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

协议类主要用于定义和封装与服务器进行通信的消息体，客户端发送消息之前都需要通过协议类将消息进行格式化，再发送给服务器。

Protocol.java

```
package util;

public class Protocol
{
    public Protocol()
    {

    }

    public String getRegisterInformation(String information)
    {
        String instruction = "Register:";
        String massage = instruction + information;
        return massage;
    }

    public String getLogInInformation(String information)
    {
        String instruction = "LogIn:";
        String massage = instruction + information;
        return massage;
    }

    public String getToFriendmassage(String information)
    {
        String instruction = "SendMassage:";
        String massage = instruction + information ;
        return massage;
    }
}
```